Geometric Problems in Cartographic Networks

Geometrische Problemen in Kartografische Netwerken
(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad
van doctor aan de Universiteit Utrecht
op gezag van de Rector Magnificus, Prof. Dr. W.H. Gispen,
ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen
op maandag 29 maart 2004 des middags te 16:15 uur

door

Sergio Cabello Justo

geboren op 1 december 1977,
te Lleida

Promotor: Prof. Dr. Mark H. Overmars
Copromotor: Dr. Marc J. van Kreveld
Faculteit Wiskunde en Informatica
Universiteit Utrecht

Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 99.
This work has been supported mainly by the Cornelis Lely Stichting, and partially by a Marie Curie Fellowship of the European Community programme "Combinatorics, Geometry, and Computation" under contract number HPMT-CT-2001-00282, and a NWO travel grant.

Dankwoord / Agradecimientos

It is my wish to start giving my deepest thanks to my supervisor Marc van Kreveld and my promotor Mark Overmars. I am also very grateful to Mark de Berg, who also participated in my supervision in the beginning. Without trying it, you made me think that I was in the best place to do my thesis.

I would also like to thank Pankaj Agarwal, Mark de Berg, Jan van Leeuwen, Günter Rote, and Dorothea Wagner for taking part in the reading committee and giving comments. I would also like to thank the co-authors who directly participate in this thesis: Mark de Berg, Erik Demaine, Steven van Dijk, Yuanxin (Leo) Liu, Andrea Mantler, Günter Rote, Jack Snoeyink, Tycho Strijk, and, of course, Marc van Kreveld.

I would also like to thank all the colleagues at the Institute that shared some time with me, and also those in Berlin. Han-Wen deserves special gratitude because he hardly complained after hearing the same song for the twentieth time. He also made that my PhD time would be more enjoyable and helped finishing this thesis with the samenvatting.

Mirando hacia atrás, hay varias personas que se merecen mi más sincero agradecimiento. En Joan Gimbert em va contagiar la primera il·lusió per les matemàtiques. Pepe Bordeta y Enrique fueron buenos profesores, mucho más de lo que en su momento pude ver. Carles Padró em va fer la primera approximació a la recerca. I en Ferran Hurtado, qui ja va fer que m'enamorés de la geometria computacional poc abans d'entrar a l'universitat.

También me gustaría dar gracias a mis amigos por eso, por ser mis amigos.

Y como lo más dulce siempre se deja para el final, me gustaría acabar dando gracias a mi familia por estar siempre ahí. Hvala tudi moji mravlji.

Ljubljana, febrero de 2004.

Contents

Chapter 1

Introduction

Visualization of data is a basic and important topic; it helps to analyze or extract information from the data, as well as to communicate it. If we restrict ourselves to spatial or geographical data, then we are talking about *cartography*, and, in particular, about the generation of maps.

The design of a map is a very complex task. A cartographer cannot just project the data onto a piece of paper, but he has to worry about its readability. The way to improve the quality of the map is by using so-called *white lies*. For example, in a road map of Europe, if the thickness of a road would be proportional to its width in real world, the user would not notice it on the map. Therefore, the cartographer needs to make it thicker on the map than it would be otherwise according to the map scale.

Furthermore, the design of a map is not only a complex task, but it also involves subjective decisions. For example, the cartographer has to decide what information is not relevant and can be omitted from the map, how to improve its readability in cluttered areas, where to put labels with relevant features, what legend to use, and so on. "How to Lie with Maps", by Monmonier [103], is a classical, nice-to-read book on how these decisions affect the map.

1.1 Automated cartography

The appearance of computers in the 20th century has affected many fields, and cartography has not been kept aside of this revolution, leading to the so-called automated cartography research field. Initially the topics consisted of automating certain tasks originally done by cartographers; later, the area mixed with the research on geographic information systems.

Automated generalization is, probably, the most recurrent topic within automated cartography. According to Heywood et al. [87], *generalization* is "the process by which information is selectively removed from a map in order to simplify pattern without distortion of overall content". Another relevant topic is automated *labelling* of a map, that is, placing labels with the entities of a map,

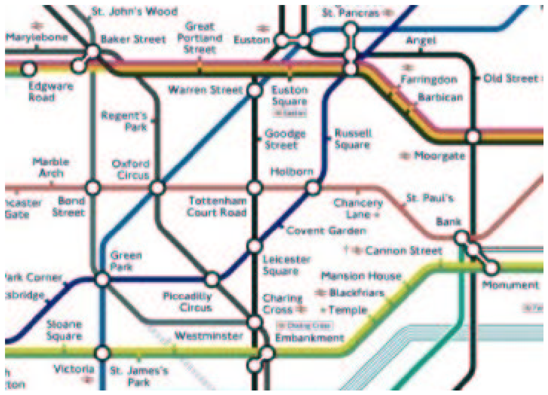


Figure 1.1: Detail of the underground map of London, a classical example of a schematic map.

such as names of cities or rivers. These two processes are done so often during the design of a map that automating them saves hours of work.

In recent years, the concept of interactive maps and maps on demand, that is, maps that are tailored to the user's wishes or necessities, are getting an increased interest, mostly due to the widespread use of Internet. These maps include route maps based on queries as a special case. Given that there is a lot of demand for such maps, their construction has to be fully automated.

One of the types of maps that allow for automated construction is the schematized map, which inspired most of the research presented in this thesis. In a *schematic map*, a set of nodes and their connections are displayed in a highly simplified form, since the precise shape of the connections and position of the nodes is not so important; see Figure 1.1. To preserve the recognizability for map readers, the approximate layout must be maintained, however.

Cartograms are another interesting type of map that gives rise to challenging computational problems; see Figure 1.2. A *cartogram* is a map in which the size of each entity is proportional to some value associated with the entity [35, Chapter 14][47, Chapter 10]. *Area cartograms* are the most common example, in which the area of each region is proportional to some function of the region, like for example, its population. In *linear cartograms*, we want to display a network in such a way that the length of a connection is related to some characteristic of the connection. In ordinary maps, this length is correlated (through a planar projection of the sphere) to the length of the connection in the real world. However, we may be interested in showing, for instance, the travel time for each connection, or the amount of traffic on each connection. Part of this thesis is concerned with linear cartograms.

We have analyzed some of the steps, or considerations, that cartographers face when designing schematic maps and linear cartograms. We have abstracted them, and converted them into mathematically formulated computational prob-

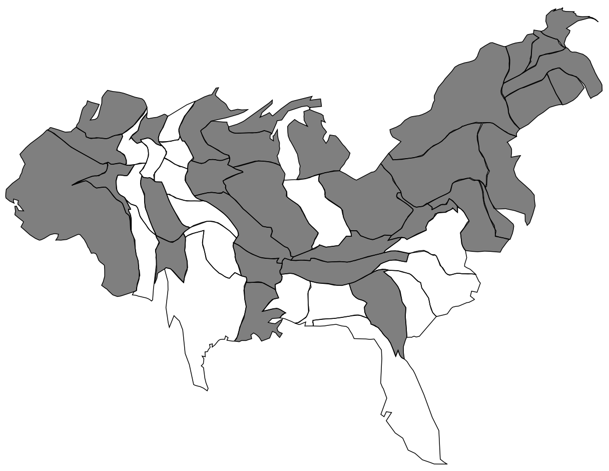


Figure 1.2: Cartogram of the United States based on the electoral votes in the 1992 presidential elections (from Edelsbrunner and Waupotitsch [61]).

lems. The abstraction process is fundamental to be able to deal with the problems in the context of computational geometry. Furthermore, this allows that the considered problems find applications not only in cartography, but also in robotics, visualization of data, and graph drawing, to name a few other research areas.

We want to stress that our research is, by no means, trying to completely solve the problem of automatically generating maps, but is aimed at providing tools that will successfully perform specific tasks that cartographers may find useful when designing a cartographic network. This is the main purpose of the research contained in this thesis.

In the following sections, we describe the context in which the results of this thesis are embedded: computational geometry. Then, we discuss the related work that has been done. At the end of this chapter, we give a thesis overview, explaining the computational problems that are analyzed in subsequent chapters, and discussing their motivation within cartography.

1.2 Computational geometry

We have considered the problems from a computational geometry perspective. *Computational geometry* is the branch of algorithmics that deals with problems with a strong geometric flavour. Generally, the problems are considered in constant-dimensional spaces. In fact, most of the research has been done in two or three dimensions because of the numerous applications.

In computational geometry, effort is put into designing efficient algorithms, where efficiency is measured both in the asymptotic running time and in the required memory space.

The field started in the seventies, and the first books on the topic were by Edelsbrunner [58], and by Preparata and Shamos [110]. The book by de Berg et al. [45] contains (more than) the appropriate background you need to understand this thesis. A more programming-oriented book of computational geometry is by O'Rourke [108], and a more discrete and combinatorial slant can be found in the book by Matoušek [100].

Some of the problems that we considered are instances of *geometric optimization* problems, that is, optimization problems with an essential geometric component. In this context, the concepts of approximation algorithms and approximation schemes play an important role. A good introduction to these concepts with a geometric flavour is by Bern and Eppstein [20]. The surveys about geometric optimization by Agarwal and Sharir [6], and by Arora [10] are also relevant.

Besides cartography and geographical information systems, the field of computational geometry also shares interests with other research areas: data structures, motion planning, virtual environments, computational biology, graph drawing, discrete and combinatorial geometry, computer graphics, computer vision, shape matching and recognition, computational topology, and many more.

1.3 Related work

In this section, we give an overview of some of the research that has been done in automated cartography. We start with general references, and then we concentrate on more relevant work on schematic maps and linear cartograms. In each chapter, we give relevant references for the specific problem that we consider.

A good introduction to cartography is the book by Dent [47], and a more informal one is the one by Monmonier [103]. Recent research results on automated cartography are presented at the International Cartographic Conference, Auto-Carto (Proceedings of the International Symposium on Computer-Assisted Cartography), the International Symposium on Spatial Data Handling, the Dagstuhl seminars on Computational Cartography (September 1999, May 2001, September 2003), and some other conferences.

Schematic maps have become a quite standard way to convey information, and therefore the work on this area has increased over the past years. For example, the PhD thesis by Avelar [11] and an ArcGIS Schematic package, produced by ESRI [1], have appeared recently. However, the automated construction of schematic maps has already been studied in earlier papers. Elroi [65, 66, 67] describes an approach where the paths are first simplified, then they are placed on a grid to assure restricted orientations, and then crowded areas are locally enlarged to avoid regions with too high density. No technicalities of the algorithm or running-time analysis are given in these papers. Avelar and Müller [13] describe an iterative procedure that attempts to rotate all links of an input network into one of the four main orientations. No upper bound on the running time can be given. Also, the output may contain links in other orientations than the desired ones. Barbowsky, Latecki and Richter [15] have used an iterative discrete curve evolution, based on a local measure, to simplify the curves while keeping the local spatial ordering. All these methods rely on iterative approaches where the connections should converge to the major orientations, while displacing the junctions. For this type of techniques, it is difficult or even impossible to guarantee the convergence of the procedure.

The paper by Neyer [107] describes a line-simplification algorithm where the final paths must have links in one of c given orientations only, and stay close enough to the original path. The algorithm minimizes the number k of links in the output in $O(n^2 k \log n)$ time, and when it is applied to disjoint paths, the output may have intersections, which change the topology of the map. In a paper by Raghavan et al. [112], a wiring is made by connecting pairs of points by non-intersecting, 2-link orthogonal paths. This can be seen as a schematic map where only two different schematic paths are possible for each pair of points. The problem can be solved in $O(n \log n)$ time, as shown by Imai and Asano [90], but the model is too restrictive and the relative positions in the resulting network may be different than in the original network, making the recognizability of the features harder. For depicting the schematic paths, the work by Duncan et al. [55] is also worth mentioning, where paths are redrawn in the same homotopic class and with maximal separation.

We are not aware of any research done for the construction of linear cartograms. However, there is much research done on the problem of drawing graphs with specified edge lengths. For cartographic purposes we have the natural restriction that the drawing should be planar, and then the problem has been considered by Di Battista and Vismara [52], Eades and Wormald [56], and Whitesides [120].

If we drop the planarity condition, then the problem has been studied in the fields of computational geometry [41, 69, 115, 122], rigidity theory [40, 83, 91], sensor networks [36, 114], and structural analysis of molecules [19, 42, 84]. It appears frequently when only distance information is known about a given structure, such as the atoms in a protein [19, 42, 84] or the nodes in an ad-hoc wireless network [36, 111, 114].

Other research related to schematic maps is map generalization for road networks and line simplification. However, the objectives in such problems are quite different. In general one does not consider achieving a given number of links per path and/or having restricted orientations, but instead tries to keep the main features of each path while reducing the number of coordinates to describe it. One of the most popular algorithms for simplification of paths is by Douglas and Peucker [54], but much more research has been done; see the notes by Weibel [119] for a general reference.

Also related is the research on VLSI layout design [76, 97], where the number of edges in the output is generally not considered critical, and research on graph drawing [51, 118], where the positions of the endpoints are usually not fixed. A recent, related topic that is relevant to schematization is the rendering of particular routes under queries [7, 12]. In this case, the paths are also simplified, but there is more flexibility to distort the input map because only the objects in the surroundings of the route are displayed. Another interesting research topic within cartographic networks is using graphs for displaying the train interconnection data in the railway network. This problem has been treated by Brandes et al. [23, 24].

1.4 Thesis overview

Consider a drawing of a transportation network, such as road or railway map. To design a schematic version of it, or to produce a linear cartogram, the cartographer performs actions that can be classified into three types: displace the nodes (or junctions) of the network, modify the shape of the connections, or both together. Let us analyze these actions independently.

1.4.1 Displacing nodes

Consider the actions that displace nodes, which from now on we will imagine to be points. Independent of the reason to displace a point, its new position has to be close to the original one. A natural way to abstract this is by restricting the new position of a point to be inside a fixed region around the original position, and then the problem becomes: given a collection $S_1, \ldots, S_n$ of regions in $\mathbb{R}^2$, find a “good”placement $p_1, \ldots, p_n$ with $p_i \in S_i$. There are two issues we have to handle: what is a “good” placement, and which regions are the appropriate ones?

The cartographer displaces the points with respect to their original position to improve the quality of the map under construction. So, let us assume that we have a quality function $Q : S_1 \times \cdots \times S_n \to \mathbb{R}$ that measures the quality of a placement $(p_1, \ldots, p_n) \in S_1 \times \cdots \times S_n$. We can then say that our objective would be to provide a best placement, that is, points $p_1^*, \ldots, p_n^*$, with $p_i \in S_i$ such that

$$Q(p_1^*, \ldots, p_n^*) = \max_{(p_1,\ldots,p_n)\in S_1\times\cdots\times S_n} Q(p_1, \ldots, p_n).$$

A quality function that evaluates all the aspects of the placement would be very complex, and becomes infeasible from the computational point of view. Instead, we have considered two particular quality functions that play a fundamental role in the design of schematic maps for networks.

The first quality function captures the fact that, in a schematic map, the connections between nodes are usually displayed by a small number of straight-line segments, and usually horizontal, vertical, or also diagonal (45 or 135 degrees) segments are preferable; see Figure 1.1. This leads to the problem of, given a graph on the points $p_1, \ldots, p_n$, find a placement that maximizes the number of straight-line edges that have horizontal, vertical, and perhaps diagonal orientation. Because a picture is worth a thousand words, take a look at Figure 1.3. We refer to this problem as the aligning-points problem. We show that finding a best placement under this criterion is NP-hard, and provide several approximation schemes whose performance depend on various parameters. These results, previously published in [34], are described in Chapter 2.

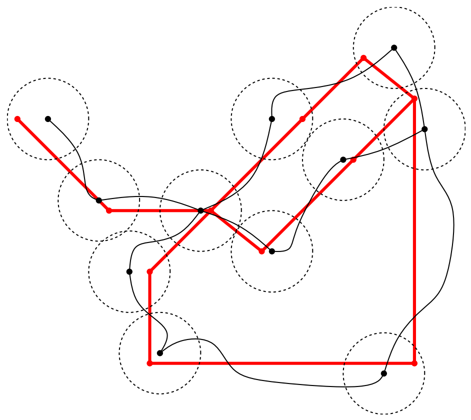


Figure 1.3: The aligning-points problem. In thin black, the original network is shown. The disk around each point represents the region where it can be moved. We want to maximize the number of edges with horizontal, vertical, or diagonal orientation. In this case, a possible optimal solution is the thick network shown.

Regarding the second quality function that we will consider, a rule of thumb tells that the readability of the map improves as the separation between its features increases. This leads to the problem of maximizing the distance between the placed points, and, in particular, the problem of maximizing the distance between any pair of points as much as possible. Take a look at Figure 1.4 to see an example. We refer to this problem as the spreading-points problem. The problem was shown to be NP-hard by Baur and Fekete [16], and Fiala, Kratochvíl, and Proskurowski [71]. When the regions are disks, we provide efficient approximation algorithms with constant-factor approximation. These results were previously published in [28], and we report them in Chapter 3.

Let us remark that in these problems, we did not deal with the actual choice of the regions, or which regions are preferable, but we assume that the regions are already given. Under this assumption, both problems belong to the area of geometric optimization.

1.4.2 Deforming connections

Let us consider one of the connections in the original map, and the corresponding connection in the schematic map. If a road passes to the North of an important

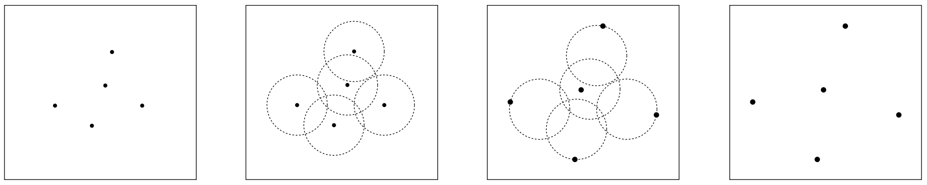


Figure 1.4: The spreading-points problem. On the left, we have a collection of points. We consider a disk around each of them (center left), and then we allow each point to be displaced within its disk. The objective is to maximize the distance of the closest pair, and for the depicted example, we would get the situation shown on the right.

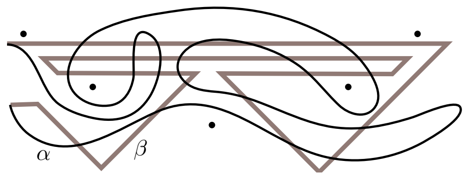


Figure 1.5: Are the paths α and β homotopic, that is, can α be continuously deformed into β without touching any of the points?

city, we do not want the schematized version to pass to the South of that city, but to keep the relative positions. If we think of the construction of the schematic path as a continuous deformation of the original path, we can formulate this requirement as follows: the original path is transformed into the schematic one in a continuous way, fixing its endpoints and without crossing any "important point", where "important point" refers to a city or any other point feature whose relative position with respect to the path we want to maintain.

Homotopy of paths is a well-known concept in topology that captures this idea. Therefore, to understand the problem and extract its features, we consider the basic decision problem: given two paths and a set of "important points" P, can we deform one path into the other one without passing over any point in P? Or, using the topological terminology, are the two paths homotopic in the plane minus P? See Figure 1.5 for an example. If both the paths and the point set P have complexity n, then we can decide it in $O(n \log n)$ time in case the paths are simple, and in $O(n^{3/2} \log n)$ time in case the paths self-intersect. Lower bounds for both cases (simple and self-intersecting paths) are also presented, and, in particular, they show that the algorithm for the simple case is asymptotically optimal. These results were previously published in [32], and we present them in Chapter 4.

Next we return to the problem of constructing a schematic map. We assume that the input is a planar embedding of a graph consisting of polygonal paths between specified points called endpoints. We are interested in producing another planar embedding where all endpoints have the same positions, and every path is displayed as a two-link or three-link path where links are restricted to certain orientations; see Figure 1.6. Furthermore, as discussed above, the output map should be equivalent to the input map in the sense that a continuous deformation exists such that no path passes over an endpoint during the transformation. For maps whose paths do not intersect, this equivalence implies that the cyclic order of paths around endpoints is maintained.

Chapter 5 contains our formalization of the problem, a computationally optimal algorithm to deal with it, and discusses the quality of the schematic maps given by our implementation. These results were previously published in [30], improving our previous works [29, 33]. Figure 1.6 has been produced by our implementation, and more examples are shown in the figures of Chapter 5, where we also discuss extensions of the algorithm.

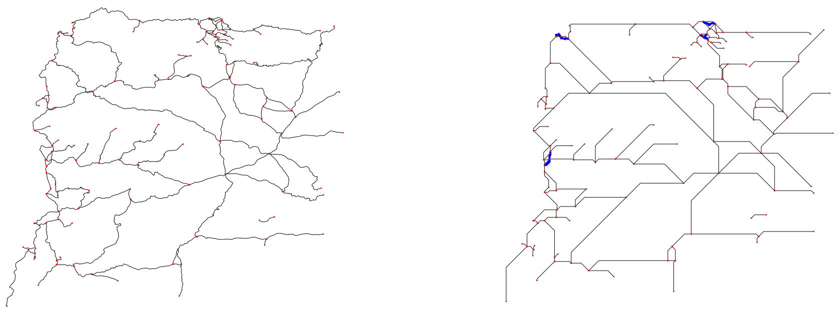


Figure 1.6: Northwest of the Iberic Peninsula. Left: the original map. Right: the schematized version made by the implementation described in Chapter 5.

1.4.3 Doing everything at the same time

To construct a schematic map by displacing points and deforming paths, both simultaneously, is too complex. Let us be more specific: so far, no algorithm provides provable results in this context. Some authors [11, 12, 13, 15, 65, 66, 67] give iterative processes that do everything at once, and that in general may give nice, pleasant results. However, there is no provable guarantee that the output will be optimal, or even good. For example, it may well happen that the algorithm cannot modify anything in the original map. More specifically, no combinatorial algorithms have been proposed that can handle connections and nodes together.

If there are no such theoretical, provable results, where is then the difficulty? The basic issue is that many simpler problems are already computationally intractable. For example, consider the problem of generating a linear cartogram.

The construction of such a map can be modeled by defining the length of each edge appropriately and trying to realize the graph with these edge lengths. So we can abstract the generation of a linear cartogram to the following natural problem: given a graph G, can we construct a planar straight-line embedding of G where the edges have a prescribed length? Observe that in real-life applications, we would also like to keep some resemblance with the original network, and so we may restrict where the vertices of the graph can be embedded. However, as we show in Chapter 6, the problem is already NP-hard without this restriction; that is, there is little hope that efficient algorithms will exist for this problem. This is true even for a very restricted class of graphs, such as 3-connected, bounded degree, and bounded face degree graphs. This problem is discussed in Chapter 6, whose contents were previously published in [31].

Chapter 2

Aligning points

This chapter presents combinatorial methods to displace the important locations or junctions of a schematic network in order to get connections that are single line segments whose orientations are prescribed to be one of a finite set of orientations.

As justified in Section 1.4.1, we consider the following abstract, geometric optimization problem: Let P be a given set of n points in the plane, and for each point $p_i \in P$ some region S_i around it. Furthermore, a graph is given of which the nodes correspond one-to-one with the points of P (and therefore with the regions). Find for each point p_i a position in its region such that the number of alignments with other points of P is maximized. Here alignment is for a given, constant number of orientations, and alignment only counts (is optimized) for two points whose nodes are connected in the graph. A formal definition of the problem is given in next section.

The rest of the chapter is organized as follows. In Section 2.1 we formalize the problem and show that if we are able to approximate the optimal solution when the graph is a tree, then we also obtain an approximation for planar graphs. In Section 2.2 we show that two rather simple versions of the problem are NP-hard. We also give an inapproximability result for general graphs provided that P$\neq$NP. In Sections 2.3, 2.4, and 2.5, we give approximation algorithms for different cases of the problem. Both the approximation factors and the time bounds depend on the properties of the regions and the set of orientations; the results are summarized in Table 2.1. More specifically, in Section 2.3 we give a polynomial time approximation scheme (PTAS) when the graph is a tree. In Section 2.4 we use the same approach to get several approximation algorithms for planar graphs. For the case of rectangular regions and horizontal and vertical orientations only, we give a polynomial time approximation scheme as well. This is based on the results from Baker [14] and is explained in Section 2.5. We finish with the conclusions and some open problems related to aligning points.

orientations	graph, region	approximation ratio	time	reference
1	tree, any region	1	$O(n^2)$	Theorem 2.13
	planar graph, any region	$\frac{1}{3}$	$O(n^2)$	Corollary 2.14
		$\frac{k-1}{k}$	$O(k(2n)^{3k+1})$	Theorem 2.21
2	tree, convex	$\frac{k}{k+1}$	$O(k9^k n^2)$	Theorem 2.11
	planar graph, convex	$\frac{k}{3(k+1)}$	$O(k9^k n^2)$	Corollary 2.12
	tree, rectangles	1	$O(n^3)$	Theorem 2.16
	planar graph, rectangles	$\frac{1}{3}$	$O(n^3)$	Corollary 2.17
		$\frac{k-1}{k}$	$O(k(2n)^{6k+1})$	Theorem 2.24
	planar graph, disjoint rectangles	$\frac{1}{3}$	$O(n^2)$	Corollary 2.15
		$\frac{k-1}{k}$	$O(k(2n)^{3k+1})$	Corollary 2.23
$h > 2$	planar graph, convex	$\frac{1}{3h}$	$O(n^2)$	Corollary 2.18
		$\frac{2k}{3h(k+1)}$	$O(k9^k n^2)$	Corollary 2.19
		$\frac{k}{3(k+1)}$	$n^{O(2^k)}$	Theorem 2.20
		$\frac{k}{h(k+1)}$	$O(k(2n)^{3k+1})$	Corollary 2.22

Table 2.1: Approximation algorithms in this chapter. k stands for an arbitrary positive integer that governs a trade-off between approximation ratios and running times.

2.1 Preliminaries

2.1.1 Formulation of the problem

In this section we formalize the problem of alignment. Recall that a graph $G = (\mathcal{S}, E)$ consists of a set of nodes $\mathcal{S}$ and a set of edges $E \subset \binom{\mathcal{S}}{2}$. In particular, we consider graphs where each node corresponds to a convex region in the plane. Given a fixed set of orientations Z, we define a function χ_Z that assigns to pairs of regions the value 1 if there is a line with orientation in Z that intersects both regions, and 0 otherwise. In particular, for two points p, q, we have $\chi_Z(p, q) := \chi_Z(\{p\}, \{q\}) = 1$ if the line through p and q has its orientation in Z, and 0 otherwise. For the application to cartography, the orientations will typically be axis-parallel ($|Z| = 2$) or also including diagonal lines (with slope 1 or -1, so $|Z| = 4$).

The problem can be stated as follows: given a set of n convex regions, $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$, a graph $G = (\mathcal{S}, E)$ on those regions, and a set of orientations Z, place n points $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ to maximize the function

$$\sum_{\{S_i, S_j\} \in E} \chi_Z(p_i, p_j).$$

We denote the maximum value by $M_Z(G)$, or simply $M(G)$, as we consider the given orientations Z to be fixed.

A $\frac{1}{r}$-approximation of $M_Z(G)$, sometimes also called an r-approximation of $M_Z(G)$, where $r \geq 1$, is a collection of n points $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ such that

$$\sum_{\{S_i, S_j\} \in E} \chi_Z(p_i, p_j) \geq \frac{1}{r} M_Z(G)$$

A polynomial time approximation scheme (PTAS) is a family $\mathcal{A}$ of approximation algorithms such that, for any $\epsilon > 0$, $\mathcal{A}$ contains a $(1 - \epsilon)$-approximation algorithm running in $O(n^{f(\epsilon)})$ time for some function f. That is, for any constant value $\epsilon > 0$, the family $\mathcal{A}$ contains a polynomial time algorithm giving a $(1 - \epsilon)$-approximation.

For our application to cartography, we usually assume G to be a planar graph. Typical regions S_i that we consider are scaled Voronoi cells, convex polygons, rectangles, and circles. However, it turns out that we only need to distinguish the case of axis-parallel rectangles and any other convex region. Regions can overlap or not, which leads to slightly different results. When the regions overlap, the placement of two points can coincide, and in this case we also assume that they are aligned.

We remark that possibly, the computed placement does not give a planar straight-line embedding. In fact we are not assuming that an embedding is given initially. If this would be the case, the new embedding may be non-equivalent to the original one.

For a region S, we define $L_Z(S)$ to be the set of lines tangent to S that have their orientation in Z (see Figure 2.1). In the algorithm to be described, we will

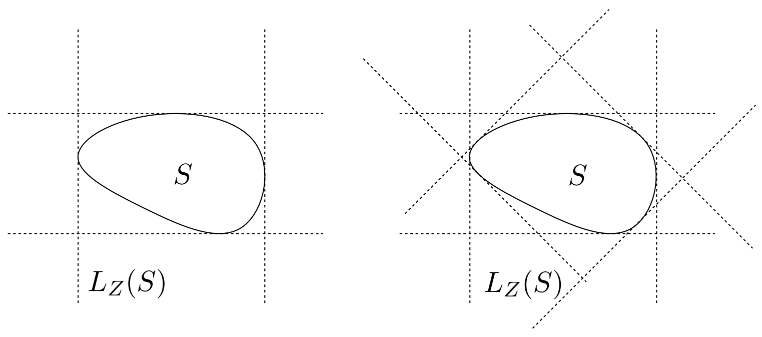


Figure 2.1: Left: $L_Z(S)$ for a region S when the orientations in Z are axis-parallel. Right: $L_Z(S)$ when the orientations in Z are axis-parallel and diagonal

subdivide region S into cells $C^1, \ldots, C^t$. We will also use the notation $L_Z(C^j)$ for the lines with orientation in Z that are tangent to the cell C^j. For a set L of lines, we will use $\mathcal{A}(L)$ for the arrangement in the plane induced by L (see [45, 58] for the concept).

2.1.2 Decomposing the original graph

It appears to be difficult to develop a general technique that gives a good approximation algorithm for any graph G, any shape of region, and any set of alignment orientations. But if G is a tree, we will present a general approach in Section 2.3 that gives several different polynomial time approximation results, depending on the shape of the regions and the number of alignment orientations. Furthermore, it is known that a planar graph G can be decomposed into three trees (or forests), such that every edge of G appears in exactly one tree (or forest) [80]. Such a partition can be found in $O(n \log n)$ time, which is the main ingredient for the following result.

Lemma 2.1 *Given $r \geq 1$, if for any tree $\mathcal{T}$ we can compute a $\frac{1}{r}$-approximation of $M_Z(\mathcal{T})$ in $O(T(n))$ time, then we can compute a $\frac{1}{3r}$-approximation of $M_Z(G)$ for any planar graph G in $O(T(n) + n \log n)$ time.*

Proof: Given a planar graph G, we decompose it into three edge disjoint forests $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$. Let A_i be a $\frac{1}{r}$-approximation of $M_Z(\mathcal{F}_i)$. The value $A := \max\{A_1, A_2, A_3\}$ can be computed in $O(n \log n + 3T(n) + 1) = O(n \log n + T(n))$ time, and it is a $\frac{1}{3r}$-approximation of $M_Z(G)$. Indeed, consider the placement $p_0, \ldots, p_{n-1}$ that achieves $M_Z(G)$. Then

$$M_Z(G) = \sum_{\{S_i, S_j\} \in E} \chi_Z(p_i, p_j) = \sum_{k=1}^{3} \Big(\sum_{\{S_i, S_j\} \in \mathcal{F}_k} \chi_Z(p_i, p_j) \Big) \leq \sum_{k=1}^{3} M_Z(\mathcal{F}_k),$$

and because A_i is a $\frac{1}{r}$-approximation of $M_Z(\mathcal{F}_i)$ we get

$$M_Z(G) \leq \sum_{k=1}^{3} M_Z(\mathcal{F}_k) \leq \sum_{k=1}^{3} rA_i \leq 3rA.$$

□

So, basically, when we approximate the original problem for the special case of trees we also obtain an approximation for a planar graph. The same approach also works for general graphs. For a given natural number k and a graph G, Gabow [73] shows how to get in $O(kn^{\frac{3}{2}}\sqrt{n + k\log n})$ time an edge-disjoint partition of G into k forests, or report that no such partition exist. However, since the minimum k that makes possible the edge-disjoint partition into forests can be $\Omega(n)$, the approximation ratio for a general graph would be $O(\frac{1}{n})$ times the one for trees, which is not really interesting.

2.2 Hardness of the problem

We show the computational hardness of two rather simple versions of the aligning problem:

- The regions are horizontal segments and we want to maximize the number of vertical alignments (so $|Z| = 1$). We call this the one-dimensional problem.
- The regions are convex, the underlying graph is a path, and we want to maximize the number of vertical and horizontal alignments. The proof is by Woeginger [121], and we call this the two-dimensional problem.

In the following we give both proofs.

2.2.1 One-dimensional problem

We reduce E3-SAT (Exact3-SATisfiability) to the one-dimensional problem to prove the hardness. The reduction implies an inapproximability result for non-planar graphs. An E3-SAT instance is a formula of t Boolean variables $x_1, \ldots, x_t$ given by m conjunctive clauses $C_1, \ldots, C_m$, where each clause contains exactly 3 literals (a variable or its negation). MAX-E3-SAT is the associated optimization problem: given an E3-SAT instance, find an assignment to the variables that maximizes the number of satisfied clauses.

Theorem 2.2 *Let Z be the vertical orientation, let $\mathcal{S} = \{S_0, \ldots S_{n-1}\}$ be a set of horizontal segments, and let $G = (\mathcal{S}, E)$ be a graph. For any $\epsilon > 0$, it is NP-hard to compute a $(\frac{15}{16} + \epsilon)$-approximation of $M_Z(G)$*

Proof: Given an E3-SAT instance ϕ with t variables $x_1, \ldots, x_t$ and m clauses $C_1, \ldots, C_m$, we construct an aligning problem $\mathcal{P}_\phi$ as follows (see Figure 2.2):

1. take $\mathcal{S} := \emptyset, E := \emptyset$;
2. for each Boolean variable x_i, add the horizontal interval $I_i := [i - \frac{1}{3}, i + \frac{1}{3}]$ to $\mathcal{S}$;
3. for each clause C_j, add the horizontal interval $J_j := [\frac{2}{3}, t + \frac{1}{3}]$ to S;
4. for each occurrence of x_i in C_j, add the horizontal interval $I_{i,j} := [i - \frac{1}{3}, i - \frac{1}{3}] = \{i - \frac{1}{3}\}$ to $\mathcal{S}$, and the edges $\{J_j, I_i\}$ and $\{J_j, I_{i,j}\}$ to E;
5. for each occurrence of the negation of x_i in C_j, add the horizontal interval $I_{i,j} := [i + \frac{1}{3}, i + \frac{1}{3}] = \{i + \frac{1}{3}\}$ to $\mathcal{S}$ and the edges $\{J_j, I_i\}$ and $\{J_j, I_{i,j}\}$ to E.

When considering a placement in this aligning problem $\mathcal{P}_\phi$, we can assume that all points have the x-coordinate in the set $C = \{1-\frac{1}{3}, 1+\frac{1}{3}, \ldots, t-\frac{1}{3}, t+\frac{1}{3}\}$. If a point has a different x-coordinate, we displace it to the largest x-coordinate value in C that is smaller than the actual value. By doing this, we cannot decrease the number of alignments: two points that were vertically aligned keep being vertically aligned because either they were not displaced or they both have been displaced to the same x-coordinate. With this assumption, we have a bijection between the Boolean assignments of the variables $x_1, \ldots, x_t$ and the placements of the points $p_1, \ldots, p_t$ with $p_i \in I_i$: x_i is true if and only if $p_i \in I_i$ is placed at $i - \frac{1}{3}$, and false if and only if $p_i \in I_i$ is placed at $i + \frac{1}{3}$.

Consider an assignment of the Boolean variables $x_1, \ldots, x_t$ and the corresponding placement of points in the regions $I_1, \ldots, I_t$. The key observation is that a clause C_j is satisfied in the assignment if and only if we can place a point in the region J_j that results in two alignments. When C_j is not satisfied, the placement of a point in the region J_j gives exactly one alignment. Therefore, we can satisfy s clauses in ϕ if and only if we can align $m + s$ pairs of points in the corresponding problem $\mathcal{P}_\phi$. In particular, for a satisfiable instance ϕ, the optimum number of alignments is $2m$.

If we have a polynomial time $(\frac{15}{16} + \epsilon)$-approximation algorithm for the aligning problem, and we use it for $\mathcal{P}_\phi$, where ϕ is a satisfiable E3-SAT instance, we would get at least

$$(\frac{15}{16} + \epsilon)2m = \frac{30m}{16} + 2\epsilon m = m + (\frac{7}{8} + 2\epsilon)m$$

alignments. But then we would have a polynomial time $(\frac{7}{8} + 2\epsilon)$-approximation algorithm for MAX-E3-SAT on satisfiable instances, which is NP-hard by Theorem 6.5 of [81]. □

To show that the problem is NP-hard when G is planar, we will reduce planar 3-SAT to the one-dimensional alignment problem. A planar 3-SAT instance is a formula with t Boolean variables $x_1, \ldots, x_t$ given in m conjunctive clauses $C_1, \ldots, C_m$, where each clause contains at most 3 literals (a variable or its negation) and such that the bipartite graph

$$G = (\{x_1, \ldots, x_t, C_1, \ldots, C_m\}, \{\{x_i, C_j\} \mid x_i \text{ or } \neg x_i \text{ is in } C_j\})$$

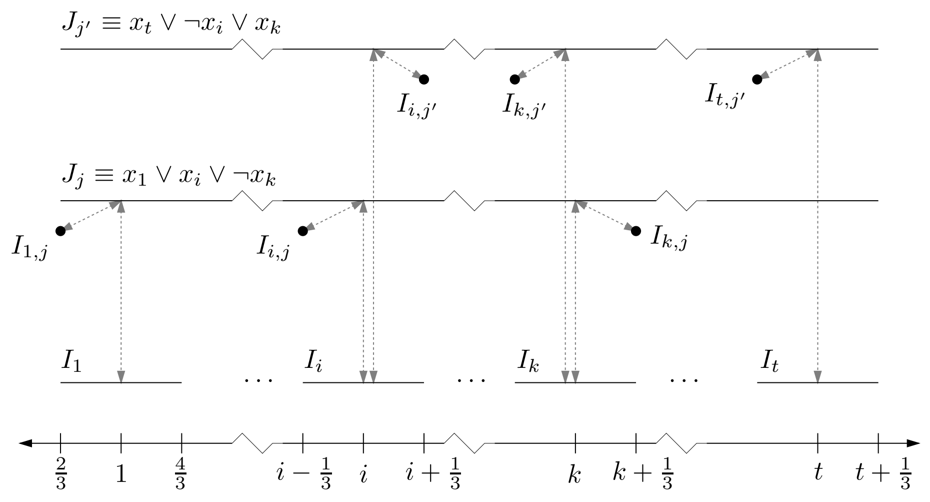


Figure 2.2: Reduction from E3-SAT to an alignment problem. The variables $x_1, \ldots, x_t$ are represented by the segments $I_1, \ldots, I_t$, and each clause C_j is represented by the segment J_j plus three segments $I_{i,j}$ that depend on its literals.

is planar (see [95]).

Corollary 2.3 *It is NP-hard to compute $M_Z(G)$ for planar graphs G.*

Proof: Consider a planar 3-SAT instance ϕ and apply the reduction used in the proof of the previous theorem to get an alignment problem $\mathcal{P}_\phi$. We claim that the graph G_ϕ of the problem $\mathcal{P}_\phi$ is planar. Observe that the nodes of the type $I_{i,j}$ have degree one, and therefore we can remove them without affecting the planarity or non-planarity of the graph. The remaining graph is

$$(\{I_1, \ldots, I_t, J_1, \ldots, J_m\}, \{\{I_i, J_j\} | \, x_i \text{ appears in } C_j\}),$$

and is isomorphic to

$$(\{x_1, \ldots, x_t, C_1, \ldots, C_m\}, \{\{x_i, C_j\} | \, x_i \text{ appears in } C_j\}),$$

which has to be planar by definition of planar 3-SAT instances.

As discussed in the previous proof, $M_Z(G_\phi) = 2m$ if and only if ϕ is satisfiable. Therefore, if for any planar graph we can compute $M(G)$, we can decide the satisfiability of planar 3-SAT instances, which is NP-hard [95]. □

It is natural to wonder if we can construct alignment problems where it is NP-complete to decide if all edges can be aligned or not. We can show that the answer is negative if we restrict ourselves to only one orientation, but hard if we there are two orientations (see Theorem 2.5).

Theorem 2.4 *Let Z be the vertical orientation, let $\mathcal{S} = \{S_0, \ldots S_{n-1}\}$ be a set convex regions and let $G = (\mathcal{S}, E)$ be a connected graph. Then $M_Z(G) = |E|$ if and only if there is a vertical line that intersects all regions.*

Proof: We can assume that the regions are horizontal segments, otherwise, we project each region onto a horizontal line and we get an equivalent problem.

If all the intervals in G are intersected by a vertical line l, then we can place the point $p_i \in S_i$ at $l \cap S_i$. It is clear that we get $M_Z(G) = |E|$ because all points are vertically aligned.

For the other implication, consider a placement $p_0, \ldots, p_{n-1}$, with $p_i \in S_i$, that achieves $M_Z(G) = |E|$ alignments. We claim that the vertical line through p_0 also goes through all other p_i. To see this, fix any S_i, and assume without loss of generality that $S_0, S_1, \ldots, S_i$ is a path in G from S_0 to S_i (it always exists because G is connected). Then, point p_0 has to be vertically aligned with point p_1, and p_1 has to be vertically aligned with p_2, and so on until p_i. Because being vertically aligned is a transitive relation, p_0 has to be vertically aligned with p_i, and both are on the same vertical line. $\square$

2.2.2 Two-dimensional problem

The following hardness proof for the two-dimensional case has been given by Woeginger [121]. The reduction is from subset-sum: given natural numbers $a_1, \ldots, a_n$ and b, does a subset $I \subset \{1, \ldots, n\}$ exists such that $\sum_{j \in I} a_j = b$? This problem is NP-complete in the weak sense [75].

Theorem 2.5 *Let Z consist of the vertical and horizontal orientations, let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of convex regions and let $G = (\mathcal{S}, E)$ be a path. It is NP-complete to decide if $M_Z(G) = |E| = n - 1$.*

Proof: Given an instance of subset-sum, we construct the following $n+2$ regions in the Euclidean plane:

- Region S_0: The origin $(0, 0)$.
- Region S_j, for $j = 1, \ldots, n$: The closed line segment with endpoints $(0, s_j)$ and $(s_j, 0)$, where $s_j = \sum_{k=1}^{j} a_k$.
- Region S_{n+1}: The point $(-1, b)$.

See Figure 2.3. The graph G is a path on the regions $S_0, \ldots, S_{n+1}$ with an edge between S_i and S_{i+1} for $i = 0, \ldots, n$. The feasible directions are horizontal and vertical.

This instance has a solution (with all $n+1$ edges in horizontal/vertical direction) if and only if the subset-sum instance has answer YES. The equivalence is immediate: If we use number a_i in the additive representation of b, then the line segment p_{i-1} to p_i is vertical. If we do not use this number, then the segment is horizontal. Hence, the y-coordinates of the path from S_i up to S_j reflects the sum of the selected subset from s_i up to s_j. $\square$

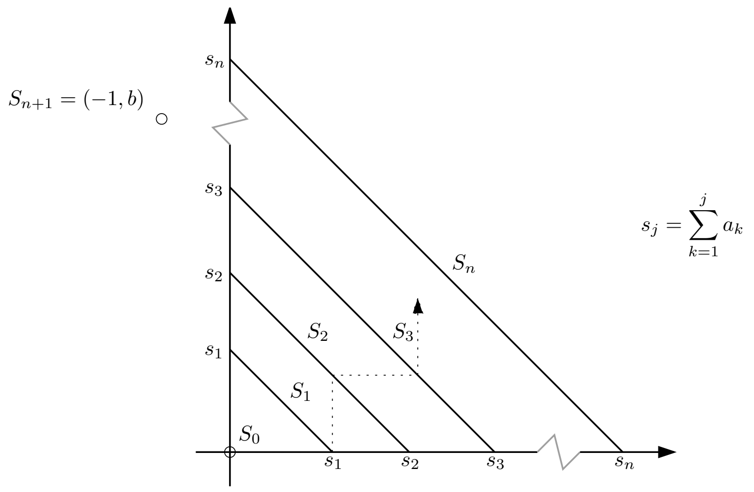


Figure 2.3: Reduction from the subset problem to the aligning problem.

2.3 The basic approach for trees

In this section we explain the algorithm for alignment for the specific case of convex regions and two aligning orientations. In the next section we will analyze what results are obtained when we apply the same technique to other versions of the alignment problem, with different region shapes and different alignment orientations.

Let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of n convex regions, and let $\mathcal{T} = (\mathcal{S}, E)$ be a tree. We choose any node, S_0, to be the root of $\mathcal{T}$. Let b_i be the complexity of the boundary of region S_i. If for an arbitrary node $S_i \neq S_0$, we remove from $\mathcal{T}$ the edge connecting S_i with its parent node, we get two subtrees. We will use $\mathcal{T}_i$ to denote the subtree containing the node S_i. We assume that nodes $S_1, \ldots, S_d$ are the neighbors of node S_0, so d is the degree of S_0. In particular, when we remove S_0 from $\mathcal{T}$, we get the subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_d$; see Figure 2.4. We use $\mathcal{T}(p_i)$ to denote the graph $\mathcal{T}$ after replacing the node S_i by p_i, that is, the point p_i is the placement chosen for the region S_i; see Figure 2.5. Fixing a point p_0 in the region S_0 makes the subproblems that appear in the subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_d$ independent, and therefore we get the following recurrence:

$$M(\mathcal{T}(p_0)) = \sum_{i=1}^{d} \max_{p_i \in S_i} \{\chi_Z(p_0, p_i) + M(\mathcal{T}_i(p_i))\}.$$

The overall idea is to subdivide (that is, partition) region S_0 into cells such that any placement within a cell will give exactly the same solution. This will be done in a recursive way: to construct the subdivision in S_0 we will use subdivisions of $S_1, \ldots, S_d$ with that same property, but only for the corresponding

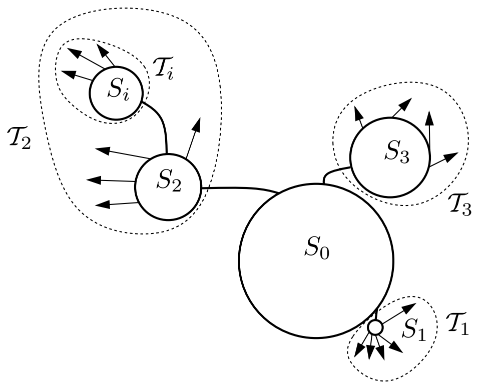


Figure 2.4: Neighbors of S_0 and the corresponding subtrees.

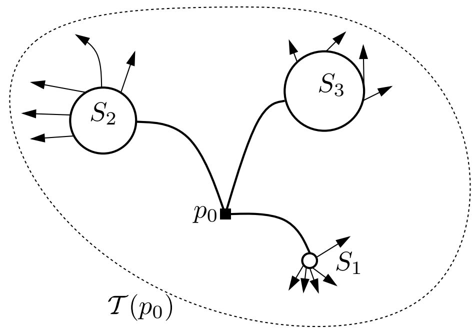


Figure 2.5: When we substitute S_0 by $\{p_0\}$ we get $\mathcal{T}(p_0)$.

subtree: each placement in a cell of S_i gives the same number of alignments in $\mathcal{T}_i$.

Definition 2.6 *A convex cell $C \subseteq S_i$ is $\mathcal{T}$-*stable *if and only if*

$$M(\mathcal{T}(p_i)) = M(\mathcal{T}(p_i')) \qquad \forall p_i, p_i' \in C.$$

We use $M(\mathcal{T}(C))$ to denote this invariant value.

It is clear that if S_i is a leaf of $\mathcal{T}$, then S_i already is a $\mathcal{T}_i$-stable cell. This gives the basis for a recursive formulation on how to make the subdivision of S_0. Let $C_i^1, \ldots, C_i^{t_i}$ be a subdivision of S_i into $\mathcal{T}_i$-stable cells. Let L_0 be the set of all lines with orientation in Z that are tangent to some cell C_i^j, where $i = 1 \ldots d$ and $j = 1 \ldots t_i$ (see the example in Figure 2.6). In other words, we have

$$L_0 = \bigcup_{i=1}^{d} \bigcup_{j=1}^{t_i} L_Z(C_i^j).$$

We can subdivide S_0 using all lines in L_0 to make an arrangement $\mathcal{A}(L_0)$ inside S_0.

Lemma 2.7 *Any cell in $\mathcal{A}(L_0) \cap S_0$ is $\mathcal{T}$-stable.*

Proof: Consider any cell C in $\mathcal{A}(L_0) \cap S_0$, and two points $p_0, p_0' \in C$. We want to show that $M(\mathcal{T}(p_0)) = M(\mathcal{T}(p_0'))$. Let the points $p_1, \dots, p_d$ with $p_i \in S_i$ be placed to attain the value $M(\mathcal{T}(p_0))$, that is,

$$M(\mathcal{T}(p_0)) = \sum_{i=1}^{d} \{\chi_Z(p_0, p_i) + M(\mathcal{T}_i(p_i))\}.$$

Let $C_i^{j_i} \subset S_i$ be the $\mathcal{T}_i$-stable cell in which p_i lies. We will show that there exist points $p_1', \dots, p_d'$ with $p_i' \in C_i^{j_i}$ such that $\chi_Z(p_0, p_i) \leq \chi_Z(p_0', p_i')$. Then we have

$$M(\mathcal{T}(p_0)) = \sum_{i=1}^{d} \{\chi_Z(p_0, p_i) + M(\mathcal{T}_i(p_i))\} \leq \sum_{i=1}^{d} \{\chi_Z(p_0', p_i') + M(\mathcal{T}_i(p_i))\}$$

and because p_i and p_i' are in the same $\mathcal{T}_i$-stable cell $C_i^{j_i}$

$$M(\mathcal{T}(p_0)) \leq \sum_{i=1}^{d} \{\chi_Z(p_0', p_i') + M(\mathcal{T}_i(p_i))\} =$$

$$= \sum_{i=1}^{d} \{\chi_Z(p_0', p_i') + M(\mathcal{T}_i(p_i'))\} \leq M(\mathcal{T}(p_0')).$$

However, by symmetry we also have $M(\mathcal{T}(p_0')) \leq M(\mathcal{T}(p_0))$ and therefore $M(\mathcal{T}(p_0)) = M(\mathcal{T}(p_0'))$.

The points $p_1', \dots, p_d'$ that we need can be found as follows. If $\chi_Z(p_0, p_i) = 0$, take $p_i' := p_i$, and the properties hold. If $\chi_Z(p_0, p_i) = 1$, let $z \in Z$ be the orientation of the line $\overline{p_0 p_i}$, and let l_i be the line through p_0' with orientation z. Because p_0 and p_0' are in the same cell C of $\mathcal{A}(L_0) \cap S_0$, the line l_i lies between the two tangents to $C_i^{j_i}$ with orientation z. Therefore, the intersection $C_i^{j_i} \cap l_i$ is nonempty, and any $p_i' \in C_i^{j_i} \cap l_i$ has the desired properties. $\square$

When we have subdivided S_0 into $\mathcal{T}$-stable cells $C_0^1, \dots C_0^{t_0}$, we can compute the maximum value $M(\mathcal{T}) = \max_{j \in \{1, \dots, t_0\}} \{M(\mathcal{T}(C_0^j))\}$. Thus, we need to be able to compute, for a $\mathcal{T}$-stable cell C, the actual number of alignments $M(\mathcal{T}(C))$: we place an arbitrary point $p_0 \in C$ and then we have

$$M(\mathcal{T}(C)) = M(\mathcal{T}(p_0)) = \sum_{i=1}^{d} \max_{j \in \{1 \dots t_i\}} \{\chi_Z(p_0, C_i^j) + M(\mathcal{T}_i(C_i^j))\}.$$

There are two important issues to address: how many cells does the subdivision of S_0 have if we recursively use Lemma 2.7, and how much time does it take to compute the value $M(\mathcal{T}(C))$ for each cell C of the subdivision. We will bound the time spent at node S_0 assuming we have already processed its children $S_1, \dots, S_d$.

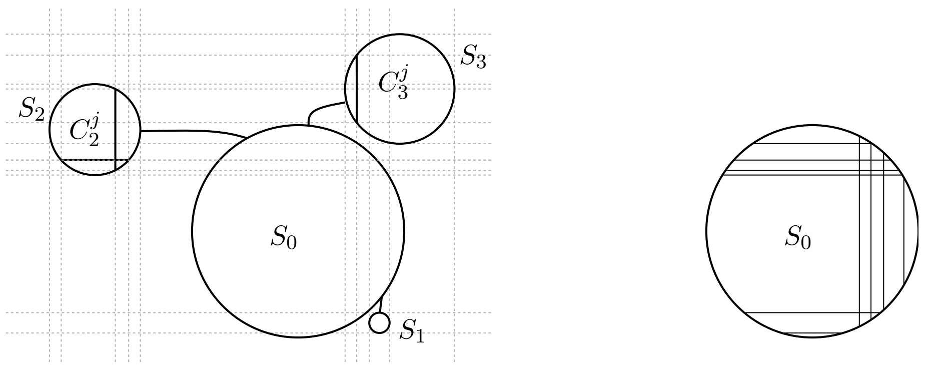


Figure 2.6: In this example S_1, S_2, S_3 are adjacent to S_0 in $\mathcal{T}$. The tangents to the cells C_i^j induce a subdivision in S_0.

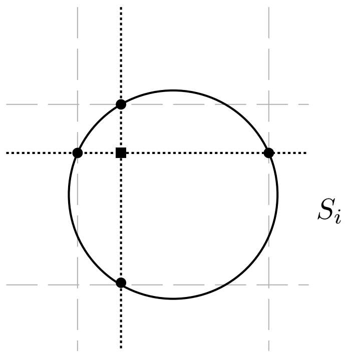


Figure 2.7: Analysis of the different tangents that can be produced. The vertex represented by a square is internal to S_i, and does not produce more tangent lines than we had. The other vertices come from an intersection of a line in L_i with the boundary of S_i, and it contributes to L_0 with at most one new line. Furthermore, the region has $2|Z| = 4$ tangents that are new lines.

Lemma 2.8 *If the tree $\mathcal{T}$ has height k, then we can subdivide S_0 into $O(9^k n^2)$ $\mathcal{T}$-stable cells in $O(9^k n^2 + b_0)$ time, where b_0 is the complexity of S_0.*

Proof: We compute the set of lines L_0 that has been used in the previous lemma, and then we compute $\mathcal{A}(L_0) \cap S_0$. We start by bounding the number of lines in L_0. Let L_i be the set of lines that are used in the recursive process to subdivide the region S_i into $\mathcal{T}_i$-stable cells. Any line in L_0 is tangent to some vertex of a cell C_i^j, where $i \in \{1, \ldots, d\}$. Three cases arise (see Figure 2.7):

- The vertex is interior to S_i, that is, it was determined by the intersection of two of the lines in L_i. Because we only have two orientations, those tangents are already present in L_i.

- The vertex is on the intersection of the boundary of S_i and a line in L_i. The region S_i is convex, so any line in L_i intersects the boundary at most twice. Therefore each line can produce at most two new lines in L_0.
- The vertex is on the boundary, but it does not lie on any line of L_i. In this case each region S_i can produce at most $2|Z| = 4$ new lines.

Therefore, we have the recursive relation $|L_0| \leq \sum_{i=1}^{d}(3|L_i| + 4)$, and we will show that $|L_0| \leq 3^{k+1}(n-1) = O(3^k n)$ by induction on the height k of the tree. Indeed, if the tree has height 0, then it consists of only one node and it is trivially true because no tangent line has been used. For the general case, observe that the set L_i of lines has been constructed recursively from the tree $\mathcal{T}_i$, which is rooted at node S_i and has height $k-1$. Therefore, if $|\mathcal{T}_i|$ denotes the number of nodes in $\mathcal{T}_i$, we have

$$|L_0| \leq \sum_{i=1}^{d}(3|L_i| + 4) \leq 4d + 3\sum_{i=1}^{d} 3^k(|\mathcal{T}_i| - 1) = 4d + 3^{k+1}\sum_{i=1}^{d}(|\mathcal{T}_i| - 1)$$

and because $\sum_{i=1}^{d} |\mathcal{T}_i| = n - 1$ and $k \geq 1$ we get

$$|L_0| \leq 4d + 3^{k+1}(n - d - 1) = 3^{k+1}(n-1) + d(4 - 3^{k+1}) \leq 3^{k+1}(n-1).$$

This finishes the inductive proof that shows $|L_0| = O(3^k n)$.

To construct L_0, we need to find, for each child S_i of S_0, the intersections of L_i with the boundary of S_i. But this has been done already when $\mathcal{A}(L_i) \cap S_i$ was computed, and therefore takes time linear in the number of lines generated. Once we have L_0, we compute $\mathcal{A}(L_0)$ and walk through the boundary of S_0 to compute $\mathcal{A}(L_0) \cap S_0$, the portion of the arrangement $\mathcal{A}(L_0)$ inside S_0. We can bound the time spent in this part by $O(b_0)$ (the complexity of the boundary of S_0) plus the complexity of the arrangement, which is $O(b_0 + (3^k n)^2) = O(9^k n^2 + b_0)$ in total. $\square$

Once we have computed all $\mathcal{T}$-stable cells $C_0^1, \ldots, C_0^{t_0}$, we can compute the values $M(\mathcal{T}(C_0^j))$. Assuming that each neighbor S_i of S_0 has been subdivided into $C_i^1, \ldots, C_i^{t_i}$ $\mathcal{T}_i$-stable cells, and that the values $M(\mathcal{T}_i(C_i^1)), \ldots, M(\mathcal{T}_i(C_i^{t_i}))$ have been computed, we will show how to compute the values of the cells in the subdivision of S_0. Observe that if we would compute for each cell C_0^j the value $M(\mathcal{T}(C_0^j))$ by examining the children, then we would spend $\Omega(d)$ time per cell, and so it would take $\Omega(9^k n^2 d)$ time. Because d can be $\Omega(n)$, this gives $\Omega(9^k n^3)$ time in the worst case. We can do better than this using a divide and conquer approach on the children of S_0.

Lemma 2.9 *We can compute $M(\mathcal{T}(C_0^1)), \ldots, M(\mathcal{T}(C_0^{t_0}))$ in $O(9^k n^2 + db_0)$ time.*

Proof: Let $T(n, d)$ be the time needed when $\mathcal{T}$ has n nodes and S_0 has d children in $\mathcal{T}$. There are two cases depending on the value of d:

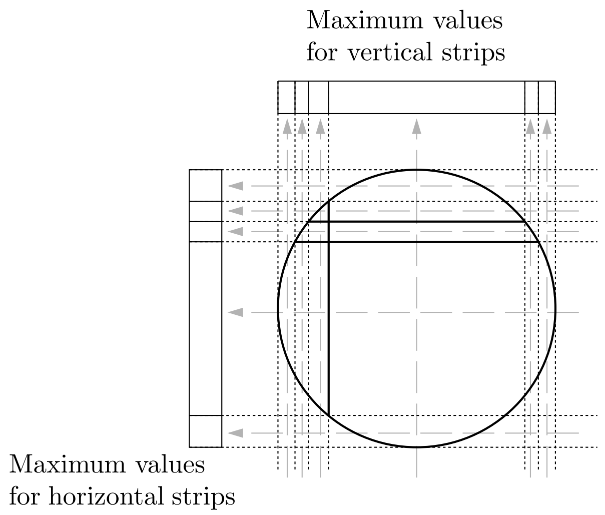


Figure 2.8: The proof of Lemma 2.9, case $d = 1$. If S_0 only has one child, we store in each strip of S_1 the maximum values.

- If $d = 1$, then S_0 has only one child, S_1. Let $C_1^1, \dots C_1^{t_1}$ be the subdivision on S_1 into $\mathcal{T}_1$-stable cells. Then, for every strip of the subdivision of S_1 with orientation in Z, we compute the maximum value $M(\mathcal{T}(C_1^j))$ over all cells C_1^j in that strip and store it in one of two arrays, one for each orientation (see Figure 2.8). We also store the maximum value over all cells $M_1 := \max\{M(C_1^1), \dots, M(C_1^{t_1})\}$. This can be done in $O(9^{k-1}n^2)$ time.

  We already had $\mathcal{A}(L_0) \cap S_0$, and now, for each cell $C_0^j \in \mathcal{A}(L_0) \cap S_0$, we take a point $p_0 \in C_0^j$:

$$M(\mathcal{T}(C_0^j)) = M(\mathcal{T}(p_0)) = \max_{j \in \{1 \dots t_1\}} \{\chi_Z(p_0, C_1^j)) + M(\mathcal{T}_1(C_1^j))\} =$$

$$= \max_{z \in Z} \Big\{ M_1, 1 + \max_{C_1^j,\, \chi_{\{z\}}(p_0, C_1^j) = 1} \{M(\mathcal{T}_1(C_1^j))\} \Big\}.$$

  But the value $\max_{C_1^j,\, \chi_{\{z\}}(p_0, C_1^j)=1}\{M(\mathcal{T}_1(C_1^j))\}$ corresponds to an entry in the array corresponding to the orientation z, so it takes constant time to compute $M(\mathcal{T}(C_0^j))$. Because $\mathcal{A}(L_0) \cap S_0$ has $O(9^k n^2)$ cells, we conclude that $T(n, 1) = O(9^k n^2)$.

- If $d > 1$, then S_0 has more than one child. In this case, we split its children into two sets $\mathcal{S}_l$ and $\mathcal{S}_r := \{S_1, \dots, S_d\} \setminus \mathcal{S}_l$, and consider the subtrees $\mathcal{T}_l$ and $\mathcal{T}_r$, where $\mathcal{T}_l$ is the connected component of $\mathcal{T} \setminus \mathcal{S}_r$ that contains S_0 and $\mathcal{T}_r$ is the connected component of $\mathcal{T} \setminus \mathcal{S}_l$ that contains S_0 (see Figure 2.9). Let $L_l \subset L_0$ be the set of lines that have been produced by nodes $S_i \in \mathcal{S}_l$ in

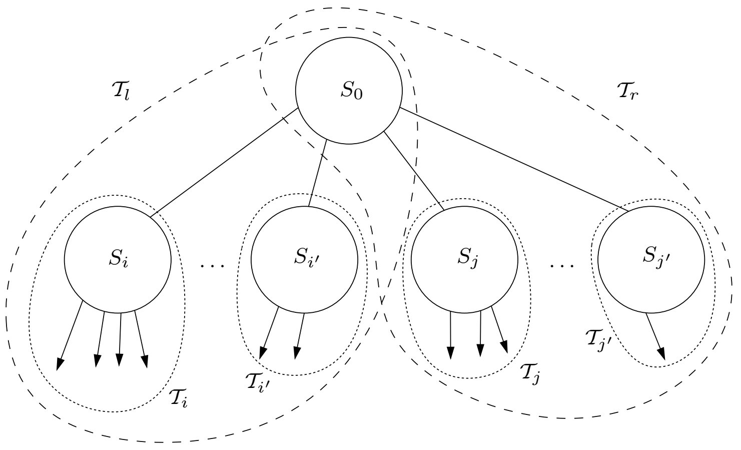


Figure 2.9: The proof of Lemma 2.9, case $d > 1$. If S_0 has more than one child we use divide and conquer.

Lemma 2.8, and let $L_r \subset L_0$ be the set of lines that have been produced by nodes $S_j \in \mathcal{S}_r$ in Lemma 2.8, thus we have $L_0 = L_l \cup L_r$. By Lemma 2.7, any cell $C_l \in \mathcal{A}(L_l) \cap S_0$ is $\mathcal{T}_l$-stable and any cell $C_r \in \mathcal{A}(L_r) \cap S_0$ is $\mathcal{T}_r$-stable. We can compute $\mathcal{A}(L_l) \cap S_0$ and $\mathcal{A}(L_r) \cap S_0$ in $O(9^k n^2 + b_0)$ time, the values $M(\mathcal{T}_l(C_l))$ for all $C_l \in \mathcal{A}(L_l) \cap S_0$ in $T(|\mathcal{T}_l|, |\mathcal{S}_l|)$ time, and the values $M(\mathcal{T}_r(C_r))$ for all $C_r \in \mathcal{A}(L_r) \cap S_0$ in $T(|\mathcal{T}_r|, |\mathcal{S}_r|)$ time. Then, because any cell $C_0^j \in \mathcal{A}(L_0) \cap S_0$ is of the form $C_l \cap C_r$, with $C_l \in \mathcal{A}(L_l) \cap S_0$ and $C_r \in \mathcal{A}(L_r) \cap S_0$, we have $M(\mathcal{T}(C_0^j)) = M(\mathcal{T}_l(C_l)) + M(\mathcal{T}_r(C_r))$, and we can compute $M(\mathcal{T}(C_0^j))$ in constant time per cell. We conclude that it takes $O(9^k n^2)$ time for all cells in $\mathcal{A}(L_0) \cap S_0$.

The two cases give the recurrence

$$T(n, d) = O(9^k n^2 + b_0) + T(|\mathcal{T}_l|, |\mathcal{S}_l|) + T(|\mathcal{T}_r|, |\mathcal{S}_r|), \qquad T(n, 1) = O(9^k n^2),$$

where we still have freedom to choose the sets $\mathcal{S}_l$ and $\mathcal{S}_r$. The choice is made as follows. Assume without loss of generality that the subtree $\mathcal{T}_1$ is the biggest among the subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_d$, that is, $|\mathcal{T}_1| \geq |\mathcal{T}_i|$ for any $2 \leq i \leq d$. We distinguish two cases depending on the size of $\mathcal{T}_1$:

- If $|\mathcal{T}_1| \geq \frac{3n}{4}$, then $\mathcal{S}_l := \{S_1\}$ and $\mathcal{S}_r := \{S_2, \ldots, S_d\}$.
- If $|\mathcal{T}_1| < \frac{3n}{4}$, we take $\mathcal{S}_l$ and $\mathcal{S}_r$ such that $\frac{n}{4} \leq |\mathcal{T}_l|, |\mathcal{T}_r| \leq \frac{3n}{4}$.

Taking $m = |\mathcal{T}_l|$ and $d' = |\mathcal{S}_l|$, we can rewrite the recurrence as

$$T(n,d) \leq \begin{cases} C 9^k n^2 & \text{if } d = 1 \\ C(9^k n^2 + b_0) + T(m, 1) + T(n - m, d - 1) & \text{if } d > 1 \text{ and } m \geq \frac{3n}{4} \\ C(9^k n^2 + b_0) + T(m, d') + T(n - m, d - d') & \text{if } d > 1 \text{ and} \\ & \frac{n}{4} < m, n - m < \frac{3n}{4} \end{cases}$$

where $C > 0$ is some fixed constant. We will show by substitution that it solves to $T(n,d) \leq 3C(9^k n^2 + (d-1)b_0) = O(9^k n^2 + db_0)$. Indeed, for the first case of the recurrence it is evident. For the second case we use $T(m,1) \leq C9^k m^2$ to get

$$\begin{aligned} T(n,d) &\leq C(9^k n^2 + b_0) + T(m,1) + T(n-m,d-1) \leq \\ &\leq C(9^k n^2 + b_0) + C9^k m^2 + 3C(9^k(n-m)^2 + (d-2)b_0) = \\ &= C9^k(n^2 + m^2 + 3(n-m)^2) + Cb_0(1 + 3(d-2)). \end{aligned}$$

Because $m \leq n$ and $n - m \leq n/4$ we have

$$T(n,d) \leq C9^k\big(n^2 + n^2 + 3(n/4)^2\big) + 3C(d-1)b_0 \leq C9^k(3n^2) + 3C(d-1)b_0.$$

For the third case we have

$$\begin{aligned} T(n,d) &\leq C(9^k n^2 + b_0) + T(m,d') + T(n-m,d-d') \leq \\ &\leq C\big(9^k n^2 + b_0 + 3(9^k m^2 + (d'-1)b_0) + 3(9^k(n-m)^2 + (d-d'-1)b_0)\big) \\ &\leq C9^k\big(n^2 + 3m^2 + 3(n-m)^2\big) + Cb_0\big(1 + 3(d'-1) + 3(d-d'-1)\big). \end{aligned}$$

Because $m^2 + (n-m)^2$ is concave, and $n/4 < m, n-m < 3n/4$, we have

$$T(n,d) \leq C9^k\big(n^2 + 3(n/4)^2 + 3(3n/4)^2\big) + 3C(d-1)b_0 \leq C9^k(3n^2) + 3C(d-1)b_0.$$

□

Putting together Lemma 2.8 and Lemma 2.9 we can show how to compute $M(\mathcal{T})$ for a tree $\mathcal{T}$ of bounded height.

Lemma 2.10 *If each region S_i has complexity $O(n)$, and $\mathcal{T} = (\mathcal{S}, E)$ has height k, we can compute in $O(9^k n^2)$ time a placement $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ that achieves $M(\mathcal{T})$ alignments.*

Proof: Starting from the region S_0, we recursively apply the subdivision done in Lemma 2.7, and for the leaves, we take the whole region as a stable cell. For the leaves S_i we take $M(\mathcal{T}_i(S_i)) := 0$. Traversing the tree $\mathcal{T}$ in a bottom-to-top fashion, for each region S_i that has been subdivided into $\mathcal{T}_i$-stable cells $C_i^1, \ldots, C_i^{t_i}$ we compute all the values $M(\mathcal{T}_i(C_i^1)), \ldots, M(\mathcal{T}_i(C_i^{t_i}))$ using Lemma 2.9. Finally, we choose a cell $C_0^{j_0}$ such that $M(\mathcal{T}(C_0^{j_0})) = \max\{M(\mathcal{T}(C_0^1)), \ldots, M(\mathcal{T}(C_0^{t_0}))\}$, and a point $p_0 \in C_0^{j_0}$. If we have kept information on how we computed $M(\mathcal{T}(C_0^{j_0}))$ in Lemma 2.9, then it is easy to find points $p_1, \ldots, p_d$ such that

$$M(\mathcal{T}(p_0)) = \sum_{i=1}^{d} (\chi_Z(p_0, p_i)) + M(\mathcal{T}_i(p_i))$$

and recursing on $M(\mathcal{T}_i(p_i))$ we get the placement for all points top-to-bottom.

Let b_i be the complexity of region S_i and let d_i be the degree of node S_i in $\mathcal{T}_i$. To bound the time needed, observe that for a node S_i that is at depth k_i, we have spent $O(9^{k-k_i}|\mathcal{T}_i|^2 + b_i)$ time to compute its subdivision into $\mathcal{T}_i$-stable cells $C_i^1, \ldots, C_i^{t_i}$ (Lemma 2.8), and $O(9^{k-k_i}|\mathcal{T}_i|^2 + d_i b_i)$ time to compute

$M(\mathcal{T}_i(C_i^1)), \ldots, M(\mathcal{T}_i(C_i^{t_i}))$ (Lemma 2.9). To bound the time of the whole process, we sum over all nodes

$$\sum_{S_i \in \mathcal{S}} O(9^{k-k_i}|\mathcal{T}_i|^2 + d_i b_i) =$$

$$= \sum_{k'=0}^{k} \Big(\sum_{S_i \text{ at depth } k'} O(9^{k-k'}|\mathcal{T}_i|^2) \Big) + \sum_{i=0}^{n-1} d_i b_i \leq$$

$$\leq \sum_{k'=0}^{k} O\Big(9^{k-k'} \sum_{S_i \text{ at depth } k'} |\mathcal{T}_i|^2\Big) + O(n) \sum_{i=0}^{n-1} d_i \leq$$

$$\leq \sum_{k'=0}^{k} O(9^{k-k'}n^2) + O(n^2) \leq O(9^k n^2).$$

□

We can combine this last result with the shifting technique of Hochbaum and Maass [88]. This technique consists of decomposing the original problem into a family of subproblems to be solved independently, while guaranteeing that the solution to at least one of the subproblems is a good approximation to the solution for the original problem. This provides a polynomial time approximation scheme (PTAS) to approximate $M(\mathcal{T})$ for any tree $\mathcal{T}$, which is the main result of this section.

Theorem 2.11 *Let Z be a set of two orientations, let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of n convex regions, each of complexity $O(n)$, and let $\mathcal{T} = (\mathcal{S}, E)$ be a tree. For any given integer $k > 0$, we can place points $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ that yield a $\frac{k}{k+1}$-approximation of $M_Z(\mathcal{T})$ in $O(k9^k n^2)$ time.*

Proof: Choose any node S_0 of $\mathcal{T}$ to be the root. We apply the shifting technique of Hochbaum and Maass [88] in order to decompose the problem into trees of height k while controlling the loss in optimality. For $u = 0, \ldots, k$, consider the forest $\mathcal{F}_u$ that is obtained by removing from $\mathcal{T}$ the parent edge from any node that has distance $u + i \cdot (k+1)$ to the root node, where i is any integer. If we root each tree in $\mathcal{F}_u$ at the node that was closest to S_0 in $\mathcal{T}$, then it has height at most k, and because $|\mathcal{F}_u| \leq n$ we can use Lemma 2.10 to determine the optimum value $M_Z(\mathcal{F}_u)$ in $O(9^k n^2)$ time. It is then clear that it takes $O(k9^k n^2)$ time to compute $M := \max\{M_Z(\mathcal{F}_0), \ldots, M_Z(\mathcal{F}_k)\}$.

We claim that M is a $\frac{k}{k+1}$-approximation of $M_Z(\mathcal{T})$. To this end, consider the placement $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ that achieves $M_Z(\mathcal{T})$ alignments. Because for each forest $\mathcal{F}_u$ we have $M_Z(\mathcal{F}_u) \geq \sum_{\{S_i,S_j\} \in \mathcal{F}_u} \chi_Z(p_i, p_j)$, then

$$(k+1)M \geq \sum_{u=0}^{k} M_Z(\mathcal{F}_u) \geq \sum_{u=0}^{k} \sum_{\{S_i,S_j\} \in \mathcal{F}_u} \chi_Z(p_i, p_j).$$

But if an edge is not in $\mathcal{F}_t$, then it is present in all $\mathcal{F}_u$ with $u \neq t$, and so each edge of $\mathcal{T}$ appears exactly k times in the sum. This means that

$$\sum_{u=0}^{k} \sum_{\{S_i,S_j\}\in\mathcal{F}_u} \chi_Z(p_i,p_j) \geq k \sum_{\{S_i,S_j\}\in\mathcal{T}} \chi_Z(p_i,p_j) = kM_Z(\mathcal{T}),$$

and we conclude that M is a $\frac{k}{k+1}$-approximation of $M_Z(\mathcal{T})$. □

Corollary 2.12 *Under the assumptions of the previous theorem, if $G = (\mathcal{S}, E)$ is a planar graph and k a given positive integer, we can compute a $\frac{k}{3(k+1)}$-approximation of $M_Z(G)$ in $O(k9^k n^2)$ time.*

Proof: Combine Theorem 2.11 with Lemma 2.1. □

2.4 Specific results for planar graphs

For different settings (regions and orientations) we can apply the same idea of dividing each region S_i into cells that are stable. The same recursive idea as explained before works out, but the analysis gives different results. Reconsidering Lemmas 2.8, 2.9, and 2.10 for each setting separately will give us the new bounds. We distinguish the following cases.

2.4.1 Any region, one orientation

Theorem 2.13 *Let Z consist of one orientation, let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of n regions, and let $\mathcal{T} = (\mathcal{S}, E)$ be a tree. We can place points $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ that yield $M_Z(\mathcal{T})$ in $O(n^2)$ time.*

Proof: We assume without loss of generality that the orientation for alignment to be considered is vertical. Also, as has been noted in the proof of Theorem 2.4, we can assume that the regions are horizontal segments, otherwise, we project each region onto a horizontal line and we get an equivalent problem.

In Lemma 2.8 we can get a more tight bound for $|L_0|$: in this setting each region produces two tangents (the vertical lines through its endpoints), and those are all the tangents that are created through the process, which means $|L_0| \leq 2n$. The lines L_0 induce a partition of the interval S_0 into $O(n)$ intervals, regardless of the height of $\mathcal{T}$. For Lemma 2.9, we can compute in a straightforward way the values $M(\mathcal{T}(C))$ in $O(d)$ time per cell C, where d is, as before, the degree of S_0. This means that we can accomplish Lemma 2.9 in $O(nd)$ time. For the time bound in Lemma 2.10, we have to sum over all nodes S_i the time spent at each node. If we denote by d_i the degree of S_i at $\mathcal{T}_i$ then we use

$$\sum_{i=0}^{n-1} O(|\mathcal{T}_i| d_i) \leq O\Big(\sum_{i=0}^{n-1} nd_i\Big) = O\Big(n \sum_{i=0}^{n-1} d_i\Big) = O(n^2)$$

time to accomplish Lemma 2.10. As this is independent of the height of $\mathcal{T}$, we directly get the statement. $\square$

This, together with Lemma 2.1, leads to the following result.

Corollary 2.14 *Under the assumptions of the previous theorem, if $G = (\mathcal{S}, E)$ is a planar graph, we can get a $\frac{1}{3}$-approximation of $M_Z(G)$ in $O(n^2)$ time.*

2.4.2 Axis-parallel rectangles, axis orientations

If the regions are disjoint rectangles and the orientations are axis-parallel, the placement of a point inside the region can be done independently for each axis orientation, and we can use the results from the previous subsection.

Corollary 2.15 *Let Z be the orientations of the coordinate axes, let $\mathcal{S} = \{S_0, \dots, S_{n-1}\}$ be a set of n axis-parallel rectangles, and let $G = (\mathcal{S}, E)$ be a planar graph. If the regions $\mathcal{S}$ are disjoint, we can get a $\frac{1}{3}$-approximation of $M_Z(G)$ in $O(n^2)$ time.*

Proof: Consider the vertical orientation $z_v \in Z$, and use Corollary 2.14 to compute a placement yielding a $\frac{1}{3}$-approximation of $M_{\{z_v\}}(G)$. This placement only fixes the x-coordinates of the points, and we can independently decide the y-coordinate of each point because the regions are rectangles. The y-coordinate is computed using Corollary 2.14 to get a $\frac{1}{3}$-approximation of $M_{\{z_h\}}(G)$, where $z_h \in Z$ is the horizontal orientation. Because the regions are disjoint, no two points coincide and we have constructed a placement achieving at least $\frac{1}{3}(M_{\{z_v\}}(G) + M_{\{z_h\}}(G)) \geq \frac{1}{3}M_Z(G)$ alignments. $\square$

If the regions overlap, then this procedure only gives us a $\frac{1}{6}$-approximation because if points placed for different regions coincide, then we are counting them as two alignments. Without considering each orientation independently, but both as a whole, we can approximate this problem at the cost of another linear factor. Again, we have to consider first the case of a tree, and then combine it with Lemma 2.1 to approximate the planar graph case.

Theorem 2.16 *Let Z consist of the orientations of the coordinate axes, let $\mathcal{S} = \{S_0, \dots, S_{n-1}\}$ be a set of n axis-parallel rectangles, and let $\mathcal{T} = (\mathcal{S}, E)$ be a tree. We can place points $p_0, \dots, p_{n-1}$ with $p_i \in S_i$ that yield $M_Z(\mathcal{T})$ in $O(n^3)$ time.*

Proof: We reconsider Lemmas 2.8, 2.9, and 2.10 for this particular setting. In Lemma 2.8 we can get a more tight bound for $|L_0|$: in this setting each region produces four tangents (the axis-aligned lines containing the boundary of the region), and those are all the tangents that are created in the process, which means $|L_0| \leq 4n$. The lines L_0 induce a partition of the rectangle S_0 into $O(n^2)$ rectangles, regardless of the height of $\mathcal{T}$. For Lemma 2.9, we can compute in a straightforward way the values $M(\mathcal{T}(C))$ in $O(d)$ time per cell C, where d is, as before, the degree of S_0. This means that we can accomplish Lemma 2.9 in

$O(n^2 d)$ time. For the time bound in Lemma 2.10, we have to sum over all nodes S_i the time spent at each node. If we denote by d_i the degree of S_i at $\mathcal{T}_i$ then we use

$$\sum_{i=0}^{n-1} O(|\mathcal{T}_i|^2 d_i) \leq O\big(\sum_{i=0}^{n-1} n^2 d_i\big) = O\big(n^2 \sum_{i=0}^{n-1} d_i\big) = O(n^3)$$

time to accomplish Lemma 2.10. As this is independent of the height of $\mathcal{T}$, we directly get the statement. □

Corollary 2.17 *Under the assumptions of the previous theorem, if $G = (\mathcal{S}, E)$ is a planar graph, we can get a $\frac{1}{3}$-approximation of $M_Z(G)$ in $O(n^3)$ time.*

2.4.3 Convex regions, more than two orientations

We next assume that we are interested in alignment in $|Z| > 2$ orientations ($|Z|$ is a constant). For example, for schematic maps, alignment in the horizontal, vertical, and two diagonal orientations is important (thus $|Z| = 4$). There are different approaches that lead to different results.

Corollary 2.18 *Let $G = (\mathcal{S}, E)$ be a planar graph with n nodes. We can find a $\frac{1}{3|Z|}$-approximation of $M_Z(G)$ in $O(n^2)$ time.*

Proof: We consider $|Z|$ different problems, each one with a different orientation but with the same graph G. For each orientation $z \in Z$, we use Corollary 2.14 to compute a $\frac{1}{3}$-approximation of $M_{\{z\}}(G)$ in $O(n^2)$ time, and we take the placement achieving the maximum A over all of them. Because in the placement achieving $M_Z(G)$ alignments there is one orientation $\tilde{z} \in Z$ with at least $\frac{1}{|Z|} M_Z(G)$ alignments, then for the chosen placement we get $A \geq \frac{1}{3} M_{\{\tilde{z}\}}(G) \geq \frac{1}{3|Z|} M_Z(G)$ alignments. □

Corollary 2.19 *For any integer $k > 0$, we can find a $\frac{2k}{3(k+1)|Z|}$-approximation of $M_Z(G)$ in $O(k 9^k n^2)$ time.*

Proof: We consider $\binom{|Z|}{2}$ different problems, each one with a different pair of orientations but with the same graph G. For each pair of orientations $\{z_i, z_j\} \subset Z$, we use Corollary 2.12 to compute a $\frac{k}{3(k+1)}$-approximation of $M_{\{z_i, z_j\}}(G)$ in $O(k 9^k n^2)$ time, and we take the placement achieving the maximum A over all of them. Because in the placement achieving $M_Z(G)$ alignments there is a pair of orientations $\{\tilde{z}_i, \tilde{z}_j\} \subset Z$ with at least $\frac{2}{|Z|} M_Z(G)$ alignments, then for the chosen placement we get $A \geq \frac{k}{3(k+1)} M_{\{\tilde{z}_i, \tilde{z}_j\}}(G) \geq \frac{2k}{3(k+1)|Z|} M_Z(G)$ alignments. □

Theorem 2.20 *Let Z be a set with a constant number of orientations, let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of n convex regions, each of complexity $O(n)$, and let*

$G = (\mathcal{S}, E)$ be a planar graph. For any given integer $k > 0$, we can place points $p_0, \ldots, p_{n-1}$ with $p_i \in S_i$ that yield a $\frac{k}{3(k+1)}$-approximation of $M_Z(G)$ in $n^{O(2^k)}$ time.

Proof: We will show that for a tree $\mathcal{T} = (\mathcal{S}, E)$ of height k we can compute a $\frac{k}{k+1}$-approximation of $M_Z(\mathcal{T})$ in $n^{O(2^k)}$ time. Then the proof of Theorem 2.11 implies that we can get a $\frac{k}{3(k+1)}$-approximation of $M_Z(G)$ in $kn^{O(2^k)} \leq n^{O(2^k)+1} = n^{O(2^k)}$ time, as desired.

Let us assume that $\mathcal{T} = (\mathcal{S}, E)$ is a tree of height k, and reconsider the proofs of Lemma 2.8, Lemma 2.9, and Lemma 2.10 for this particular setting. The bound for $|L_0|$ in Lemma 2.8 is no longer true because each internal vertex produces $h-2$ additional tangent lines (when we had only two orientations this did not happen). If for each child S_i of S_0, L_i is the set of lines that is used in the recursive process to subdivide the region S_i into $\mathcal{T}_i$-stable cells, then the lines in L_0 come from intersection points of two lines in L_i, from intersection points of a line in L_i with the boundary of the region S_i, and the $2|Z|$ tangents to S_i itself. Taking $h = |Z|$, the recursion that we get is

$$|L_0| \leq \sum_{i=1}^{d} \Big((h-2)\binom{|L_i|}{2} + (2h-1)|L_i| + 2h \Big) < 2h \sum_{i=1}^{d} (|L_i|+1)^2.$$

This solves to $|L_0| \leq (2h)^{(2^k-1)} n^{(2^k)} - 1$ by induction on the height k of $\mathcal{T}$: if $k = 0$, then $n = 1$ and it holds. For $k \geq 1$ we have

$$|L_0| \leq 2h \sum_{i=1}^{d} (|L_i|+1)^2 \leq 2h \sum_{i=1}^{d} \big((2h)^{(2^{k-1}-1)} |\mathcal{T}_i|^{(2^{k-1})} \big)^2,$$

and because $\sum_{i=1}^{d} |\mathcal{T}_i| = n - 1$ we get

$$|L_0| \leq (2h)^{(2^k-1)} \sum_{i=1}^{d} |\mathcal{T}_i|^{(2^k)} \leq (2h)^{(2^k-1)} (n-1)^{(2^k)}.$$

Therefore, $|L_0| = O((2h)^{(2^k-1)} n^{(2^k)}) = n^{O(2^k)}$ because we assume that $h = |Z|$ is constant, and we can subdivide S_0 into $n^{O(2^k)}$ $\mathcal{T}$-stable cells in $O\big(n^{O(2^k)} + b_0\big) = n^{O(2^k)} + O(b_0)$ time.

Regarding Lemma 2.9, we can compute in a straightforward way the values $M(\mathcal{T}(C))$ in $O(d)$ per cell C, where d is, as before, the degree of S_0. This means that we can accomplish Lemma 2.9 in $O(d)n^{O(2^k)} = n^{O(2^k)}$ time because $d \leq n$. For the time bound in Lemma 2.10, we have to sum over all nodes S_i the time spent in each node. If we denote by d_i the degree of S_i in $\mathcal{T}_i$, and because $b_i = O(n)$, we use

$$\sum_{i=0}^{n-1} (n^{O(2^k)} + b_i) \leq n^{O(2^k)+1} = n^{O(2^k)}$$

time to accomplish Lemma 2.10. $\square$

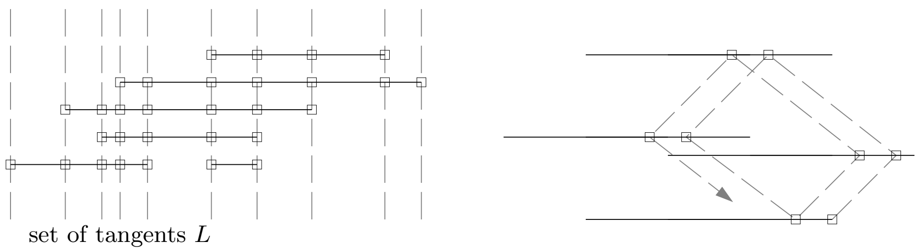


Figure 2.10: Left: Discretization used in the proof of Theorem 2.21. The boxes represent the only points to take into account. Right: If the regions are intervals and we consider two orientations to align points, we cannot discretize the problem.

2.5 Axis-aligned regions and orientations

We can use Baker's approach [14] to optimally solve k-outerplanar graphs. Combining it with the shifting technique of Hochbaum and Maass [88], we get the following polynomial time approximation scheme for planar graphs when we care about only one orientation.

Theorem 2.21 *Let Z consist of one orientation, let $\mathcal{S} = \{S_0, \dots, S_{n-1}\}$ be a set of n regions, and let $G = (\mathcal{S}, E)$ be a planar graph. For any given integer $k > 1$, we can place points $p_0, \dots, p_{n-1}$ with $p_i \in S_i$ that yield a $\frac{k-1}{k}$-approximation of $M_Z(G)$ in $O(k(2n)^{3k+1})$ time.*

Proof: As in Theorem 2.13, it is enough to consider vertical alignments and regions that are horizontal segments. The proof goes in two steps. First, we show that for any k-outerplanar graph G, we can find a placement of points that attains the optimal solution $M_Z(G)$ in $O((2n)^{3k+1})$ time. Second, we will show how this leads to the theorem.

Let L be the set of vertical lines going through the endpoints of the segments. Consider for each segment S_i the set of points $\tilde{S}_i := S_i \cap L$. Because L contains at most $2n$ vertical lines, $\tilde{S}_i$ consists of at most $2n$ points (see Figure 2.10, left). Now, instead of considering to place the point p_i anywhere in S_i, we want to place it at some point of $\tilde{S}_i$. In other words, if $\tilde{G}$ is the graph G, where each node S_i is replaced by $\tilde{S}_i$ (the graphs are isomorphic, but the nodes represent different sets), we have $M(G) = M(\tilde{G})$. Now that we have discretized the problem we can use Baker's approach.

Consider the slice boundaries and the slices as defined by Baker [14] (we will follow her notation). In a level i slice boundary, we have at most $(2n)^i$ different ways of placing the points in the corresponding segments. Thus, for each level i slice, we can encode the maximum over all possible placements in its boundary using a table with at most $(2n)^{2i}$ entries. The operations between the tables are straightforward, and the most expensive one is merging two level

i slices that share some level i boundary: it takes $O((2n)^{3i})$ time. If the graph is k-outerplanar, we have $i \leq k$, and we have to perform $O(n)$ operations with the tables. This concludes the first part of the proof.

For the given planar graph G and the integer $k > 0$, consider the graph G_u that we get by removing the edges connecting any level $u+ki$ vertex with a level $u+ki+1$ vertex, for all integers i. This graph G_u is composed of k-outerplanar graphs, so we can find the best placement of points as shown before. By the pigeon hole principle, there is some $u \in \{0, \ldots, k-1\}$ such that $M_Z(G_u)$ is at least $\frac{k-1}{k} M_Z(G)$. Computing all $M_Z(G_u)$, for $u = 0, \ldots, k-1$, and taking the maximum leads to the result. □

Corollary 2.22 *For any set of orientations Z, and for general regions, we can get a $\frac{k-1}{|Z|k}$-approximation of $M_Z(G)$ in $O(k(2n)^{3k+1})$ time.*

Proof: Like in the proof of Corollary 2.18, we can solve each orientation independently using the previous theorem and take the maximum over all of them. The result follows. □

As noticed in the proof of Corollary 2.15, if the regions are disjoint rectangles the placement of a point inside the region can be done independently for each axis orientation and we get the following result.

Corollary 2.23 *Let Z be the orientations of the coordinate axes, let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of n axis-parallel rectangles, and let $G = (\mathcal{S}, E)$ be a planar graph. If the regions $\mathcal{S}$ are disjoint, for any integer $k > 1$ we can get a $\frac{k-1}{k}$-approximation of $M_Z(G)$ in $O(k(2n)^{3k+1})$ time.*

When the rectangles overlap, we may be counting two alignments if two points belonging to different regions are placed exactly at the same position. We can avoid this by adapting the proof of Theorem 2.21 to rectangles.

Theorem 2.24 *Let Z consist of the orientations of the coordinate axes, let $\mathcal{S} = \{S_0, \ldots, S_{n-1}\}$ be a set of n axis-parallel rectangles, and let $G = (\mathcal{S}, E)$ be a planar graph. For any integer $k > 1$ we can get a $\frac{k-1}{k}$-approximation of $M_Z(G)$ in $O(k(2n)^{6k+1})$ time.*

Proof: The proof is identical to the one of Theorem 2.21, but now, for solving a k-outerplanar graph we discretize the whole rectangles combining the vertical and horizontal lines (or tangents). This discretization uses $(2n)^2$ points per rectangle, and thus we can do it, by the same arguments as in that proof, in $O(k(2n)^{6k+1})$ time. □

For general regions or orientations, it does not seem easy to extend this approach. The problem is that we cannot discretize the problem as we have done before: each tangent can produce more candidate points, from which we have to trace new tangents, and this process does not converge (see Figure 2.10, right).

2.6 Concluding remarks

In this chapter we have described algorithms to align points, each of which can be placed freely in their own specified region. We showed that the problem is computationally hard, and gave several approximation algorithms and approximation schemes which apply to different variations of the problem. Variations included the alignment orientations of interest, the shape of the regions, whether overlap is present in the input for any two regions, and perhaps most important, a graph on the points that specifies which alignments count in the optimization. Our results apply to trees and planar graphs, and remain valid if the edges of the graph are weighted. When the underlying graph is a tree, we have seen that the problem is NP-hard and we have provided a matching PTAS. The problems and solutions gave rise to an interesting combination of geometry and graphs.

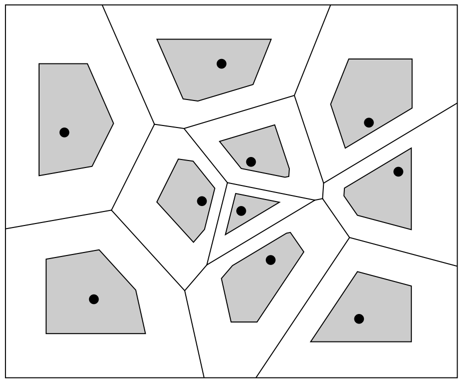


Figure 2.11: Voronoi diagram of points and scaled Voronoi cells.

Although we did not investigate the choice of the regions around each point, which surely leads to appealing algorithmical problems, we would like to make some natural considerations and remarks. A fixed, maximum allowed displacement gives rise to a fixed radius disk around each point. However, because the preservation of the approximate East-West positioning and North-South positioning is more important than for any other direction, we could instead choose squares or rectangles. Since the relative positioning with respect to points in the neighborhood is important, one could also choose to allow each point to be placed anywhere in its Voronoi cell, or in a scaled-down copy of it; see Figure 2.11. This allows points further away from other points to be displaced more than points in a cluster, a behavior that is desirable. The Delaunay triangulation of a point set is often considered to capture relevant, intuitive features of a point set. In this sense, the work by Abellanas et al. [3] is relevant. They compute the maximum perturbation that a point set can permit while keeping the same Delaunay triangulation. This maximum perturbation leads to regions that are circles, and that guarantee that the original point set and the new one have the same Delaunay triangulation.

There is room to improve the results that we have presented. In particular, more tight results for the case of planar graphs, general regions, and general orientations would be a nice improvement. Another interesting case for further research is the alignment problem with general graphs and only the vertical orientation.

Chapter 3

Spreading points

This chapter presents combinatorial methods to increase the readability in cluttered areas of the map by displacing its points in order to increase the distance between close points.

As justified in Section 1.4.1, we consider the following abstract, geometric problem: place n points, each one inside its own, prespecified disk, with the objective of maximizing the distance between the closest pair of them. The disks can overlap and have different sizes. A formal definition of the problem is given in Section 3.1, where we also discuss the related and previous work. The problem is NP-hard and does not admit a polynomial time approximation scheme (PTAS) [16, 71].

Besides the application in cartography that was discussed in the introduction, this geometric optimization problem is also relevant for graph drawing [51], and more generally in data visualization, where the readability of the displayed data is also a basic requirement. For example, when displaying a molecule in the plane, the exact position of each atom is not known, but instead, we have a region where each atom is located. In this case, we also have some freedom where to draw each atom, and a rule of thumb tells that the drawing of the molecule improves as the separation between the atoms increases.

Our results for this problem are summarized in Table 3.1. The main idea of our approach is to consider an "approximate-decision" problem in the L_∞ metric. This "approximate-decision" can be done in $O(n\sqrt{n}\log n)$ time using the data structure by Mortensen [104] and the technique by Efrat et al. [62] for computing a matching in geometric settings. Details are explained in Section 3.2.

We then combine the "approximate-decision" algorithm with the geometric features of our problem to get a 2-approximation in the L_∞ metric. This can be achieved by paying an extra logarithmic factor; see Section 3.3.

The same techniques can be used in the L_2 metric, but the approximation ratio becomes $8/3$ and the running time increases to $O(n^2)$. However, when we restrict ourselves to congruent disks, a trivial adaptation of the techniques gives an approximation ratio of ~ 2.2393. This is explained in Section 3.4. We

metric	regions	approximation ratio	running time
L_∞	arbitrary disks	2	$O(n\sqrt{n}\log^2 n)$
L_2	arbitrary disks	$\frac{8}{3}$	$O(n^2)$
	congruent disks	~ 2.2393	$O(n^2)$

Table 3.1: Approximation algorithms for spreading points in the plane.

conclude in Section 3.5 with possible directions to extend our results.

3.1 Problem formulation and related work

A precise formulation of the problem is as follows. Given a distance d in the plane, consider the function $D : (\mathbb{R}^2)^n \to \mathbb{R}$ that gives the distance between a closest pair of n points

$$D(p_1, \dots, p_n) = \min_{i \neq j} d(p_i, p_j).$$

Let $\mathcal{B} = \{B_1, \dots, B_n\}$ be a collection of (possibly intersecting) disks in $\mathbb{R}^2$ under the metric d. A *feasible* solution is a placement of points $p_1, \dots, p_n$ with $p_i \in B_i$. We are interested in a feasible placement $p_1^*, \dots, p_n^*$ that maximizes D

$$D(p_1^*, \dots, p_n^*) = \max_{(p_1,\dots,p_n) \in B_1 \times \dots \times B_n} D(p_1, \dots, p_n).$$

We use $D(\mathcal{B})$ to denote this optimal value.

A t-approximation, with $t \geq 1$, is a feasible placement $p_1, \dots, p_n$, with $t \cdot D(p_1, \dots, p_n) \geq D(\mathcal{B})$. We will use $B(p, r)$ to denote the disk of radius r centered at p. Recall that under the L_∞ metric, $B(p, r)$ is an axis-aligned square centered at p and side length $2r$. We assume that the disk B_i is centered at c_i and has radius r_i, so $B_i = B(c_i, r_i)$.

This problem is a particular instance of the problem of distant representatives, recently introduced by Fiala et al. [71, 72]: given a collection of subsets of a metric space and a value $\delta > 0$, we want a representative of each subset such any two representatives are at least δ apart. They introduced this problem as a variation of the problem of systems of disjoint representatives in hypergraphs [8]. It generalizes the problem of systems of distinct representatives, and it has applications in areas such as scheduling or radio frequency (or channel) assignment to avoid interferences.

The decision version associated to our optimization problem is the original distant representatives problem. Fiala et al. [71, 72] showed that this decision problem is NP-hard in the Euclidean and Manhattan metrics. Furthermore, their result can be modified to show that, unless $NP = P$, there is a certain constant $T > 1$ such that no T-approximation is possible. They also notice that the one-dimensional problem can be solved using the scheduling algorithm by Simons [117].

Closely related are *geometric dispersion* problems: we are given a polygonal region of the plane and we want to place n points in it such that the closest pair has the largest possible distance. This problem has been considered by Baur and Fekete [16] (see also [37, 70]), where both inapproximability results and approximation algorithms are presented. Their NP-hardness proof and inapproximability results can easily be adapted to show inapproximability results for our problem, showing also that no polynomial time approximation scheme is possible, unless $P = NP$.

3.2 A placement algorithm in L_∞

Consider an instance $\mathcal{B} = \{B_1, \dots, B_n\}$ of the problem in the L_∞ metric, and let $\delta^* = D(\mathcal{B})$ be the maximum value that a feasible placement can attain. We will consider the "approximate-placement" problem that follows: given a value δ, we provide a feasible placement $p_1, \dots, p_n$ such that, if $\delta \leq \frac{1}{2}\delta^*$ then $D(p_1, \dots, p_n) \geq \delta$, and otherwise there is no guarantee on the placement. We start by presenting an algorithm and discussing its approximation performance. Then we discuss a more efficient version of it.

3.2.1 The algorithm and its approximation ratio

Let $\Lambda = \mathbb{Z}^2$, that is, the lattice $\Lambda = \{(a, b) \mid a, b \in \mathbb{Z}\}$. For any $\delta \in \mathbb{R}$ and any point $p = (p_x, p_y) \in \mathbb{R}^2$, we define $\delta p = (\delta p_x, \delta p_y)$ and $\delta\Lambda = \{\delta p \mid p \in \Lambda\}$. Observe that $\delta\Lambda$ is also a lattice. The reason to use this notation is that we can use $p \in \Lambda$ to refer to $\delta p \in \delta\Lambda$ for different values of δ. An *edge* of the lattice $\delta\Lambda$ is a horizontal or vertical segment joining two points of $\delta\Lambda$ at distance δ. The edges of $\delta\Lambda$ divide the plane into squares of side length δ, which we call the *cells* of $\delta\Lambda$.

The idea is that whenever $2\delta \leq \delta^*$, the lattice points $\delta\Lambda$ almost provide a solution. However, we have to treat as a special case the disks with no lattice point inside. More precisely, let $Q \subset \delta\Lambda$ be the set of points that cannot be considered as a possible placement because there is another already placed point too near by. Initially, we have $Q = \emptyset$. If a disk B_i does not contain any point from the lattice, there are two possibilities:

- B_i is contained in a cell C of $\delta\Lambda$; see Figure 3.1 left. In this case, place $p_i := c_i$ in the center of B_i, and remove the vertices of the cell C from the set of possible placements for the other disks, that is, add them to Q.
- B_i intersects an edge E of $\delta\Lambda$; see Figure 3.1 right. In this case, choose p_i on $E \cap B_i$, and remove the vertices of the edge E from the set of possible placements for the other disks, that is, add them to Q.

We are left with disks, say $B_1, \dots, B_k$, that have some lattice points inside. Consider for each such disk B_i the set of points $P_i := B_i \cap (\delta\Lambda \setminus Q)$ as candidates

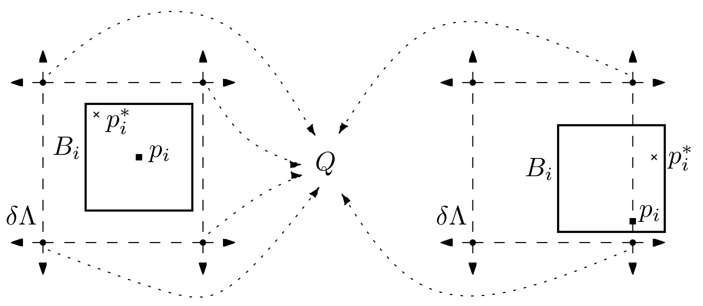


Figure 3.1: Special cases where the disk B_i does not contain any lattice point. Left: B_i is fully contained in a cell of $\delta\Lambda$. Right: B_i intersects an edge of $\delta\Lambda$.

for the placement corresponding to B_i. Observe that P_i may be empty if $(B_i \cap \delta\Lambda) \subset Q$. We want to make sure that each disk B_i gets a point from P_i, and that each point gets assigned to at most one disk B_i. We deal with this by constructing a bipartite graph G_δ with $B := \{B_1, \ldots, B_k\}$ as one class of nodes and $P := P_1 \cup \cdots \cup P_k$ as the other class, and with an edge between $B_i \in B$ and $p \in P$ whenever $p \in P_i$.

It is clear that a perfect matching in G_δ provides a feasible placement. When a matching is not possible, the algorithm reports a feasible placement by placing each point in the center of its disk. We call this algorithm PLACEMENT, and its pseudocode is given in Algorithm 1. See Figure 3.2 for an example.

Algorithm 1 PLACEMENT(δ)

```
$Q := \emptyset$ {$Q \equiv$ Lattice points that cannot be used further}
**for** all $B_i$ s.t. $B_i \cap \delta\Lambda = \emptyset$ **do**
  **if** $B_i \cap E \neq \emptyset$ for some edge $E$ of $\delta\Lambda$ **then**
    choose $p_i$ on $B_i \cap E$;
    add the vertices of $E$ to $Q$
  **else** {$B_i$ is fully contained in a cell $C$ of $\delta\Lambda$}
    $p_i := c_i$;
    add the vertices of $C$ to $Q$;
$P := \emptyset$;
**for** all $B_i$ s.t. $B_i \cap \delta\Lambda \neq \emptyset$ **do**
  $P_i := B_i \cap (\delta\Lambda \setminus Q)$;
  $P := P \cup P_i$;
construct $G_\delta := (\{B_i \mid B_i \cap \delta\Lambda \neq \emptyset\} \cup P, \{(B_i, p) | p \in P_i\})$;
**if** $G_\delta$ has a perfect matching **then**
  for each disk $B_i$, let $p_i$ be the point that it is matched to;
**else**
  for each disk $B_i$, let $p_i := c_i$;
```

In any case, PLACEMENT always gives a feasible placement $p_1, \ldots, p_n$, and

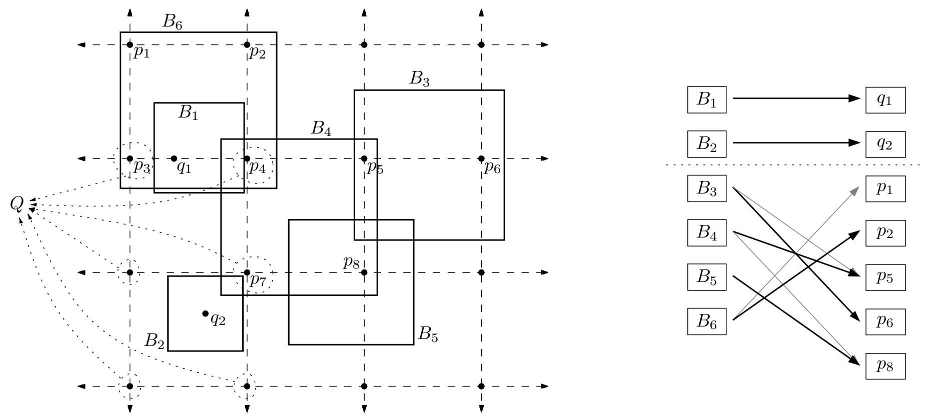


Figure 3.2: Example showing the main features of the placement algorithm in L_∞.

we can then compute the value $D(p_1, \dots, p_n)$ by finding a closest pair in the placement. We will show that, if $2\delta \leq \delta^*$, a matching exists in G_δ and moreover PLACEMENT(δ) gives a placement whose closest pair is at distance at least δ. In particular, this implies that if $B_i \cap \delta\Lambda \neq \emptyset$ but $P_i = B_i \cap (\delta\Lambda \setminus Q) = \emptyset$, then there is no matching in G_δ because the node B_i has no edges, and so we can conclude that $2\delta > \delta^*$. We first make the following definitions.

Definition 3.1 *In the L_∞ metric, we say that* PLACEMENT(δ) succeeds *if the computed placement $p_1, \dots, p_n$ satisfies $D(p_1, \dots, p_n) \geq \delta$. Otherwise, we say that* PLACEMENT(δ) fails.

Lemma 3.2 *If $2\delta \leq \delta^*$, then* PLACEMENT(δ) *succeeds.*

Proof: The proof is divided in two steps. Firstly, we will show that if $2\delta \leq \delta^*$ then the graph G_δ has a matching. Secondly, we will see that if $p_1, \dots, p_n$ is a placement computed by PLACEMENT(δ) when $2\delta \leq \delta^*$, then $D(p_1, \dots, p_n) \geq \delta$.

Consider an optimal placement $p_1^*, \dots, p_n^*$. The points that we added to Q due to a disk B_i are in the interior of $B(p_i^*, \delta^*/2)$ because of the following analysis:

- If $B_i \cap \delta\Lambda = \emptyset$ and B_i is completely contained in a cell C of $\delta\Lambda$, then p_i^* is in C, and $C \subset B(p_i^*, \delta) \subset B(p_i^*, \delta^*/2)$; see Figure 3.1 left.

- If $B_i \cap \delta\Lambda = \emptyset$ and there is an edge E of $\delta\Lambda$ such that $B_i \cap E \neq \emptyset$, then $E \subset B(p_i^*, \delta) \subset B(p_i^*, \delta^*/2)$; see Figure 3.1 right.

The interiors of the disks (in L_∞) $B(p_i^*, \delta^*/2)$ are disjoint, and we can use them to construct a matching in G_δ as follows. If $B_i \cap \delta\Lambda \neq \emptyset$, then $B(p_i^*, \delta^*/2) \cap B_i$ contains some lattice point $p_i \in \delta\Lambda$. Because the interiors of the disks $B(p_i^*, \delta^*/2)$ are disjoint, we have $p_i \notin Q$ and $p_i \in P_i$. We cannot directly add the edge (B_i, p_i) to the matching that we are constructing because it may happen that p_i is on the boundary of $B(p_i^*, \delta^*/2) \cap B_i$, but also on the boundary of $B(p_j^*, \delta^*/2) \cap B_j$. However, in this case, $B(p_i^*, \delta^*/2) \cap B_i \cap \delta\Lambda$ contains an edge of $\delta\Lambda$ inside. If we match each B_i to the lexicographically smallest point in $B(p_i^*, \delta^*/2) \cap B_i \cap \delta\Lambda$, then, because the interiors of disks $B(p_i^*, \delta^*/2)$ are disjoint, each point is claimed by at most one disk. This proves the existence of a matching in G_δ provided that $2\delta \leq \delta^*$.

For the second part of the proof, let p_i, p_j be a pair of points computed by Placement(δ). We want to show that p_i, p_j are at distance at least δ. If both were computed by the matching in G_δ, they both are different points in $\delta\Lambda$, and so they have distance at least δ. If p_i was not placed on a point of $\delta\Lambda$ (at c_i or on an edge of $\delta\Lambda$), then the lattice points closer than δ to p_i were included in Q, and so the distance to any p_j placed during the matching of G_δ is at least δ. If both p_i, p_j were not placed on a point of $\delta\Lambda$, then B_i, B_j do not contain any point from $\delta\Lambda$, and therefore $r_i, r_j < \delta/2$. Two cases arise:

- If both B_i, B_j do not intersect an edge of $\delta\Lambda$, by the triangle inequality we have $d(p_i, p_j) \geq d(p_i^*, p_j^*) - d(p_i, p_i^*) - d(p_j, p_j^*) > \delta^* - \delta/2 - \delta/2 \geq \delta$, provided that $2\delta \leq \delta^*$.
- If one of the disks, say B_i, intersects an edge E of $\delta\Lambda$, then B_i is contained in the two cells of $\delta\Lambda$ that have E as an edge. Let C be the six cells of $\delta\Lambda$ that share a vertex with E. If B_j does not intersect any edge of $\delta\Lambda$, then $B_j \cap C = \emptyset$ because otherwise $d(p_i^*, p_j^*) < 2\delta$, and so $d(p_i, p_j) \geq \delta$. If B_j intersects an edge E' of $\delta\Lambda$, we have $E \cap E' = \emptyset$ because otherwise $d(p_i^*, p_j^*) < 2\delta$. It follows that $d(p_i, p_j) \geq \delta$.

□

Notice that, in particular, if r_{min} is the radius of the smallest disk and we set $\delta = (r_{min}/\sqrt{n})$, then the nodes of type B_i in G_δ have degree n, and there is always a matching. This implies that $\delta^* = \Omega(r_{min}/\sqrt{n})$.

Observe also that whether Placement fails or succeeds is not a monotone property. That is, there may be values $\delta_1 < \delta_2 < \delta_3$ such that both Placement(δ_1) and Placement(δ_3) succeed, but Placement(δ_2) fails. This happens because for values $\delta \in (\frac{\delta^*}{2}, \delta^*]$, we do not have any guarantee on what Placement(δ) does.

The following observations will be used later.

Observation 3.3 *If* Placement(δ) *succeeds for $\mathcal{B}$, but* Placement(δ) *fails for a translation of $\mathcal{B}$, then $\delta \leq \delta^* < 2\delta$ and we have a 2-approximation.*

Observation 3.4 *If for $\delta > \delta'$,* Placement(δ) *succeeds, but* Placement(δ') *fails, then $\delta^* < 2\delta' < 2\delta$ and we have a 2-approximation.*

The algorithm can be adapted to compute Placement($\delta+\epsilon$) for an infinitesimal $\epsilon > 0$ because only the points of $\delta\Lambda$ lying on the boundaries of $B_1, \ldots, B_n$ are affected. More precisely, if a point $\delta p \in \delta\Lambda$ is in the interior of B_i, then, for a sufficiently small $\epsilon > 0$, the point $(\delta+\epsilon)p \in (\delta+\epsilon)\Lambda$ is in B_i as well. On the other hand, if a point $\delta p \in \delta\Lambda$ is on the boundary of B_i, then, for a sufficiently small $\epsilon > 0$, the point $(\delta+\epsilon)p \in (\delta+\epsilon)\Lambda$ is outside B_i if and only if δp is the point of the segment $l_p \cap B_i$ furthest from the origin, where l_p is the line passing through the origin and p. Similar arguments apply for deciding if a disk B_i is contained in a cell of $(\delta+\epsilon)\Lambda$ or it intersects some of its edges. Therefore, for an infinitesimal $\epsilon > 0$, we can decide if Placement$(\delta+\epsilon)$ succeeds or fails. This leads to the following observation.

Observation 3.5 *If* Placement(δ) *succeeds, but* Placement$(\delta+\epsilon)$ *fails for an infinitesimal $\epsilon > 0$, then $\delta^* \leq 2\delta$ and we have a 2-approximation.*

3.2.2 Efficiency of the algorithm

The algorithm Placement, as stated so far, is not strongly polynomial because the sets $P_i = B_i \cap (\delta\Lambda \setminus Q)$ can have arbitrarily many points, depending on the value δ. However, when P_i has more than n points, we can just take any n of them. This is so because a node B_i with degree at least n is never a problem for the matching: if $G_\delta \setminus B_i$ does not have a matching, then G_δ does not have it either; if $G_\delta \setminus B_i$ has a matching M, then at most $n-1$ nodes from the class P participate in M, and one of the n edges leaving B_i has to go to a node in P that is not in M, and this edge can be added to M to get a matching in G_δ.

For a disk B_i we can decide in constant time if it contains some point from the lattice $\delta\Lambda$: we round its center c_i to the closest point p of the lattice, and depending on whether p belongs to B_i or not, we decide. Each disk B_i adds at most 4 points to Q, and so $|Q| \leq 4n$. We can construct Q and remove repetitions in $O(n \log n)$ time.

If a disk B_i has radius bigger than $3\delta\sqrt{n}$, then it contains more than $5n$ lattice points, that is, $|B_i \cap \delta\Lambda| > 5n$. Because Q contains at most $4n$ points, P_i has more than n points. Therefore, we can shrink the disks with radius bigger than $3\delta\sqrt{n}$ to disks of radius exactly $3\delta\sqrt{n}$, and this does not affect to the construction of the matching. We can then assume that each disk $B_i \in \mathcal{B}$ has radius $O(\delta\sqrt{n})$. In this case, each B_i contains at most $O(n)$ points of $\delta\Lambda$, and so the set $P = \bigcup_i P_i$ has $O(n^2)$ elements.

In fact, we only need to consider a set P with $O(n\sqrt{n})$ points. The idea is to divide the disks $\mathcal{B}$ into two groups: the disks that intersect more than $\sqrt{n}$ other disks, and the ones that intersect less than $\sqrt{n}$ other disks. For the former group, we can see that they bring $O(n\sqrt{n})$ points in total to P. As for the latter group, we only need to consider $O(\sqrt{n})$ points per disk.

Lemma 3.6 *It is sufficient to consider a set P with $O(n\sqrt{n})$ points. Moreover, we can construct such a set P in $O(n\sqrt{n}\log n)$ time.*

Proof: As mentioned above, we can assume that all disks in $\mathcal{B}$ have radius $O(\delta\sqrt{n})$. Among those disks, let $\mathcal{B}_<$ be the set of disks that intersect less than $\sqrt{n}$ other disks in $\mathcal{B}$, and let $\mathcal{B}_>$ be the set of disks that intersect at least $\sqrt{n}$ other disks in $\mathcal{B}$. We treat $\mathcal{B}_<$ and $\mathcal{B}_>$ independently. We first show that for the disks in $\mathcal{B}_<$ we only need to consider $O(n\sqrt{n})$ points, and then we show that the disks in $\mathcal{B}_>$ add at most $O(n\sqrt{n})$ points to P.

For each disk $B_i \in \mathcal{B}_<$, it is enough if P_i consists of $\sqrt{n}$ points. This is so because then the node B_i is never a problem for the matching in G_δ. If $G_\delta \setminus B_i$ does not have a matching, then G_δ does not have it either. If $G_\delta \setminus B_i$ has a matching M, then at most $\sqrt{n}-1$ nodes of P_i participate in M because only the disks that intersect B_i can use a point in P_i, and there are at most $\sqrt{n}-1$ by definition of $\mathcal{B}_<$. Therefore, one of the $\sqrt{n}$ edges leaving B_i has to go to a node in P_i that is not in M, and this edge can be added to M to get a matching in G_δ.

We can construct the sets P_i for all the disks in $\mathcal{B}_<$ in $O(n\sqrt{n}\log n)$ time. First, construct Q and preprocess it to decide in $O(\log n)$ time if a query point is in Q or not. This takes $O(n \log n)$ time. For each disk $B_i \in \mathcal{B}_<$, pick points in $B_i \cap (\delta\Lambda \setminus Q)$ as follows. Initialize $P_i = \emptyset$. Take a point $p \in B_i \cap \delta\Lambda$ and check in $O(\log n)$ time if $p \in Q$. If $p \in Q$, then take another point p and repeat the test. If $p \notin Q$, then add p to P_i. Stop when P_i has $\sqrt{n}$ points or there are no points left in $B_i \cap \delta\Lambda$.

For a disk B_i we may spend $\Omega(n)$ time if, for example, $Q \subset (B_i \cap \delta\Lambda)$. However, each point in Q has appeared in the construction of at most $\sqrt{n}$ different sets P_i, as otherwise there is a point $q \in Q$ that intersects $\sqrt{n}$ disks in $\mathcal{B}_<$, which is impossible. Therefore, we have spent $O(n\sqrt{n}\log n)$ time overall.

As for the disks in $\mathcal{B}_>$, let $U = \bigcup_{B_i \in \mathcal{B}_<} B_i$ be the region that they cover. We will see how to compute $U \cap \delta\Lambda$ in $O(n\sqrt{n}\log n)$ time, and this will finish the proof. Consider the disk $B_i \in \mathcal{B}$ with biggest radius, say r, and grow each disk in $\mathcal{B}$ to have radius r. We keep calling them $\mathcal{B}$. Construct a subset $\tilde{\mathcal{B}}_> \subset \mathcal{B}_>$ as follows. Initially set $\tilde{\mathcal{B}}_> = \emptyset$, and for each $B_i \in \mathcal{B}_>$, add B_i to $\tilde{\mathcal{B}}_>$ if and only if B_i does not intersect any disk in the current $\tilde{\mathcal{B}}_>$.

Consider the number I of intersections between elements of $\tilde{\mathcal{B}}_>$ and $\mathcal{B}$. On the one hand, each disk in $\tilde{\mathcal{B}}_>$ intersects at least $\sqrt{n}$ elements of $\mathcal{B}$ by definition of $\mathcal{B}_>$, so we have $|\tilde{\mathcal{B}}_>|\sqrt{n} \le I$. On the other hand, because the disks in $\tilde{\mathcal{B}}_>$ are disjoint by construction and all the disks in $\mathcal{B}$ have the same size after the growing, each disk of $\mathcal{B}$ can intersect at most four other disks of $\tilde{\mathcal{B}}_>$, and we get $I \le 4n$. We conclude that $|\tilde{\mathcal{B}}_>| \le O(\sqrt{n})$.

Each disk in $\mathcal{B}_>$ intersects some disk in $\tilde{\mathcal{B}}_>$. Therefore, because r is at least the radius of the largest disk in $\mathcal{B}_>$, we can cover the whole region U by putting disks of radius $3r$ centered at the disks of $\tilde{\mathcal{B}}_>$. Formally, we have that $U \subset \bigcup_{B_i \in \tilde{\mathcal{B}}_>} B(c_i, 3r) =: \tilde{U}$. There are $O(\sqrt{n})$ such disks, and each of them contains $O(n)$ points of $\delta\Lambda$ because $3r = O(\delta\sqrt{n})$. We can then compute all the lattice points $\tilde{P} = \tilde{U} \cap \delta\Lambda$ in this region and remove repetitions in $O(n\sqrt{n}\log n)$ time. In particular, we have that $|U \cap \delta\Lambda| \le |\tilde{P}| = O(n\sqrt{n})$.

To report $U \cap \delta\Lambda$, we first compute U and decide for each point in $\tilde{P}$ if it belongs to U or not. Because the disks behave like pseudo-disks, U has linear size description, and we can compute it in near-linear time [92]. We can then process U to decide in $O(\log n)$ time if it contains a query point or not. We query with the $O(n\sqrt{n})$ points in $\tilde{P}$, and add to P those that are contained in

U. This accomplishes the computation of $U \cap \delta\Lambda$ in $O(n\sqrt{n}\log n)$ time. $\square$

We are left with the following problem: given a set P of $O(n\sqrt{n})$ points, and a set $\mathcal{B}$ of n disks, find a maximum matching between P and $\mathcal{B}$ such that a point is matched to a disk that contains it. We also know that each B_i contains at most $O(n)$ points of P.

If we forget about the geometry of the problem, we have a bipartite graph G_δ whose smallest class has n vertices and $O(n^2)$ edges. We can construct G_δ explicitly in $O(n^2)$ time, and then compute a maximum matching in $O(\sqrt{n}n^2) = O(n^{2.5})$ time [89]; see also [116]. In fact, to achieve this running time it would be easier to forget Lemma 3.6, and construct each set P_i by choosing $5n$ points per disk B_i, and then removing Q from them.

However, the graph G_δ does not need to be constructed explicitly because its edges are implicitly represented by the the disk-point containment. This type of matching problem, when both sets have the same cardinality, has been considered by Efrat et al. [62, 63]. Although in our setting one of the sets may be much larger than the other one, we can make minor modifications to the algorithm in [62] and use Mortensen's data structure [104] to get the following result.

Lemma 3.7 *In the L_∞ metric, the algorithm* PLACEMENT *can be adapted to run in $O(n\sqrt{n}\log n)$ time.*

Proof: We compute the set P of Lemma 3.6 in $O(n\sqrt{n}\log n)$ time, and then apply the idea by Efrat et al. [62] to compute the matching; see also [63]. The maximum matching has cardinality at most n, and then the Dinitz's matching algorithm finishes in $O(\sqrt{n})$ phases [89]; see also [116].

In each of the phases, we need a data structure for the points P that supports point deletions and witness queries with squares (disks in L_∞). If we construct the data structure anew in each phase, and P has $\Omega(n\sqrt{n})$ points, then we would need $\Omega(n\sqrt{n})$ time per phase, which is too much. Instead, we construct the data structure $\mathcal{D}(P)$ of [104] only once, and reuse it for all phases. The data structure $\mathcal{D}(P)$ can be constructed in $O(n\sqrt{n}\log n)$ time, and it supports insertions and deletions in $O(\log n)$ time per operation. Moreover, $\mathcal{D}(P)$ can be modified for answering witness queries in $O(\log n)$ time [105]: for a query rectangle R, it reports a witness point in $R \cap P$, or the empty set.

We show how a phase of the algorithm can be implemented in $O(n\log n)$ time. Consider the construction of the layered graph $\mathcal{L}$, as in Section 3 of [62]; $\mathcal{B}$ for the odd layers, and P for the even layers. We make the following modifications:

- We construct the whole layered graph $\mathcal{L}$ but without the last layer. Call it $\mathcal{L}'$. The reason is that the graph $\mathcal{L}'$ only has $O(n)$ vertices. All odd layers together have at most n vertices; an odd layer is a subset of $\mathcal{B}$, and each $B_i \in \mathcal{B}$ appears in at most one layer. In all the even layers together except for the last, the number of vertices is bounded by the matching, and so it has $O(n)$ vertices (points).

The last layer may have a superlinear number of vertices (points), but we can avoid its complete construction: if we are constructing a layer L_{2j} and we detect that it contains more than n vertices, then L_{2j} necessarily has an exposed vertex, that is, a vertex that is not used in the current matching. In this case we just put back into $\mathcal{D}$ all the vertices of L_{2j} that we already computed.

For constructing $\mathcal{L}'$ we need to query $O(n)$ times the data structure $\mathcal{D}$, and make $O(n)$ deletions. This takes $O(n \log n)$ time. If $P' \subset P$ is the subset of points that are in $\mathcal{L}'$, the final status of $\mathcal{D}$ is equivalent, in time bounds, to $\mathcal{D}(P \setminus P')$.

- For computing the augmenting paths, we use the reduced version $\mathcal{L}'$ that we have computed, together with the data structure $\mathcal{D}(P \setminus P')$. All the layers but the last can be accessed using $\mathcal{L}'$; when we need information of the last layer, we can get the relevant information by querying $\mathcal{D}(P \setminus P')$ for a witness and delete the witness element from it. We need at most one such query per augmenting path, and so we make at most n witness queries and deletions in $\mathcal{D}$. The required time is $O(n \log n)$.

- Instead of constructing the data structure $\mathcal{D}(P)$ anew at the beginning of each phase, we reconstruct it at the end of each phase. Observe that we have deleted $O(n)$ points from $\mathcal{D}(P)$. We can insert all of them back in $O(n \log n)$ time because the data structure is fully-dynamic. In fact, because the points that we are inserting back are exactly all the points that were deleted, a data structure supporting only deletions could also do the job: for each deletion we keep track of the operations that have been done and now we do them backwards.

We have $O(\sqrt{n})$ phases, and each phase takes $O(n \log n)$ time. Therefore, we only need $O(n\sqrt{n}\log n)$ time for all the phases after P and $\mathcal{D}(P)$ are constructed. $\square$

Computing the closest pair in a set of n points can be done in $O(n \log n)$ time, and so the time to decide if Placement(δ) succeeds or fails is dominated by the time needed to compute Placement(δ).

3.3 Approximation algorithms for L_∞

When we have a lower and an upper bound on the optimum value $\delta^* = D(\mathcal{B})$, we can use Lemma 3.7 to perform a binary search on a value δ such that Placement(δ) succeeds, but Placement($\delta + \epsilon$) fails, where $\epsilon > 0$ is any constant fixed a priori. Due to Lemma 3.2, this means that $\delta \leq \delta^* < 2(\delta + \epsilon)$ and so we can get arbitrarily close, in absolute error, to a 2-approximation of δ^*.

We can also apply Megiddo's parametric search [101] to find a value $\tilde{\delta}$ such that Placement($\tilde{\delta}$) succeeds, but Placement($\tilde{\delta}+\epsilon$) fails for an infinitesimally small $\epsilon > 0$. Such a value $\tilde{\delta}$ can be computed in $O(n^3 \log^2 n)$ time, and it is a 2-approximation because of Observation 3.5. Megiddo's ideas [102] of using a parallel algorithms to speed up parametric search are not very fruitful in this case because the known algorithms for computing maximum matchings [77] in parallel machines do not have an appropriate tradeoff between the number of processors and the running time.

Instead, we will use the geometric characteristics of our problem to find a 2-approximation $\tilde{\delta}$ in $O(n\sqrt{n}\log^2 n)$ time. The idea is to consider for which values δ the algorithm changes its behavior, and use it to narrow down the interval where $\tilde{\delta}$ can lie. More specifically, we will use the following facts in a top-bottom fashion:

- For a given δ, the disks B_i with radius above $3\delta\sqrt{n}$ are shrunk. Therefore, for values δ below and above $\frac{r_i}{3\sqrt{n}}$ the algorithm may construct non-isomorphic graphs G_δ.
- The disks B_i with radius $r_i < \frac{\delta^*}{4}$ are disjoint.
- If all the disks in $\mathcal{B}$ are disjoint, placing each point in the center of its disk gives a 2-approximation.
- For a value δ, assume that the disks $\mathcal{B}$ can be partitioned into two sets $\mathcal{B}_1, \mathcal{B}_2$ such that the distance between any disk in $\mathcal{B}_1$ and any disk in $\mathcal{B}_2$ is bigger than δ. If $2\delta \le \delta^*$, then we can compute a successful placement by putting together PLACEMENT(δ) for $\mathcal{B}_1$ and and PLACEMENT(δ) for $\mathcal{B}_2$.
- If for a given δ and $\mathcal{B}$ we cannot apply the division of the previous item, and each disk $B_i \in \mathcal{B}$ has radius at most R, then $\mathcal{B}$ can be enclosed in a disk B of radius $O(|\mathcal{B}|R)$.

We show how to solve this last type of problems, and then we use it to prove our main result.

Lemma 3.8 *Let $\mathcal{B}$ be an instance consisting of m disks such that each disk $B_i \in \mathcal{B}$ has radius $O(r\sqrt{k})$, and assume that there is a disk B of radius $R = O(mr\sqrt{k})$ enclosing all the disks in $\mathcal{B}$. If* PLACEMENT($\frac{r}{3\sqrt{k}}$) *succeeds, then we can compute in $O(m\sqrt{m}\log^2 mk)$ time a placement $p_1, \ldots, p_m$ with $p_i \in B_i$ that yields a 2-approximation of $D(\mathcal{B})$.*

Proof: The proof is divided into three parts. Firstly, we show that we can assume that the origin is placed at the center of the enclosing disk B. Secondly, we narrow down our search space to an interval $[\delta_1, \delta_2]$ such that PLACEMENT(δ_1) succeeds but PLACEMENT(δ_2) fails. Moreover, for any $\delta \in (\delta_1, \delta_2]$, the subset of lattice points $\tilde{P} \subset \Lambda$ such that $\delta\tilde{P}$ are inside the enclosing ball B is exactly the same. Finally, we consider all the critical values $\delta \in [\delta_1, \delta_2]$ for which the flow of control of PLACEMENT(δ) is different than for PLACEMENT($\delta + \epsilon$) or PLACEMENT($\delta - \epsilon$). The important observation is that the values δ_1, δ_2 are such that not many critical values are in the interval $[\delta_1, \delta_2]$.

Let $\mathcal{B}'$ be a translation of $\mathcal{B}$ such that the center of the enclosing disk B is at the origin. By hypothesis, $\textsc{Placement}(\frac{r}{3\sqrt{k}})$ for $\mathcal{B}$ succeeds. Therefore, if $\textsc{Placement}(\frac{r}{3\sqrt{k}})$ for $\mathcal{B}'$ fails, then $\textsc{Placement}(\frac{r}{3\sqrt{k}})$ for $\mathcal{B}$ gives a 2-approximation due to Observation 3.3, and we are done. From now on, we assume that $\textsc{Placement}(\frac{r}{3\sqrt{k}})$ succeeds and the center of B is at the origin. This finishes the first part of the proof.

As for the second part, consider the horizontal axis h. Because the enclosing disk B has radius $R = O(mr\sqrt{k})$, the lattice $(\frac{r}{3\sqrt{k}})\Lambda$ has $O(mk)$ points in $B \cap h$. Equivalently, we have $t = \max\{z \in \mathbb{Z} \text{ s.t.} (\frac{r}{3\sqrt{k}})(z, 0) \in B\} = \lfloor \frac{3R\sqrt{k}}{r} \rfloor = O(mk)$. In particular, $\frac{R}{t+1} \leq \frac{r}{3\sqrt{k}}$.

If $\textsc{Placement}(\frac{R}{t+1})$ fails, then $\textsc{Placement}(\frac{r}{3\sqrt{k}})$ is a 2-approximation due to Observation 3.4. So we can assume that $\textsc{Placement}(\frac{R}{t+1})$ succeeds. We can also assume that $\textsc{Placement}(\frac{R}{1})$ fails, as otherwise $\mathcal{B}$ consists of only one disk.

We perform a binary search in $\mathbb{Z} \cap [1, t+1]$ to find a value $t' \in \mathbb{Z}$ such that $\textsc{Placement}(\frac{R}{t'})$ succeeds but $\textsc{Placement}(\frac{R}{t'-1})$ fails. We can do this with $O(\log t) = O(\log mk)$ calls to $\textsc{Placement}$, each taking $O(m\sqrt{m}\log m)$ time due to Lemma 3.7, and we have spent $O(m\sqrt{m}\log^2 mk)$ time in total. Let $\delta_1 := \frac{R}{t'}$ and $\delta_2 := \frac{R}{t'-1}$.

Consider the lattice points $\tilde{P} := \Lambda \bigcap [-(t'-1), t'-1]^2$. For any $\delta \in (\delta_1, \delta_2]$, the points $\delta\tilde{P}$ are in B. The intuition behind why these values δ_1, δ_2 are relevant is the following. If for a point $p \in \Lambda$ we consider δp as a function of δ, then the points p that are further from the origin move quicker. Therefore, the points $\delta_2\tilde{P}$ cannot go very far from $\delta_1\Lambda$ because the extreme cases are the points on ∂B. This finishes the second part of the proof.

Before we start the third part, let us state and prove the property of δ_1, δ_2 that we will use later; see Figure 3.3. If $p \in \Lambda$ is such that $\delta_1 p$ is in the interior of B, and C_p is the union of all four cells of $\delta_1\Lambda$ having $\delta_1 p$ as a vertex, then $\delta_2 p \in C_p$, and more generally, $\delta p \in C_p$ for any $\delta \in [\delta_1, \delta_2]$. Therefore, if for a point $p \in \Lambda$ there is a $\delta \in [\delta_1, \delta_2]$ such that $\delta p \in \partial B_i$, then ∂B_i must intersect C_p.

To show that indeed this property holds, consider a point $p = (p_x, p_y) \in \Lambda$ such that $\delta_1 p$ is in the interior of B. We then have $|\delta_1 p_x| < R$, and because $|p_x| < \frac{R}{\delta_1} = \frac{R}{R/t'} = t'$ and $p_x \in \mathbb{Z}$, we conclude that $|p_x| \leq t' - 1$. This implies that

$$|\delta_2 p_x - \delta_1 p_x| = \left|\delta_1 p_x \big(\frac{\delta_2}{\delta_1} - 1\big)\right| = \left|\delta_1 p_x \big(\frac{t'}{t'-1} - 1\big)\right| = \delta_1 \frac{|p_x|}{t'-1} \leq \delta_1.$$

The same arguments shows that

$$|\delta_2 p_y - \delta_1 p_y| \leq \delta_1.$$

Since each coordinate of $\delta_2 p$ differs by at most δ_1 of the coordinates of $\delta_1 p$, we see that indeed $\delta_2 p$ is in the cells C_p of $\delta_1\Lambda$.

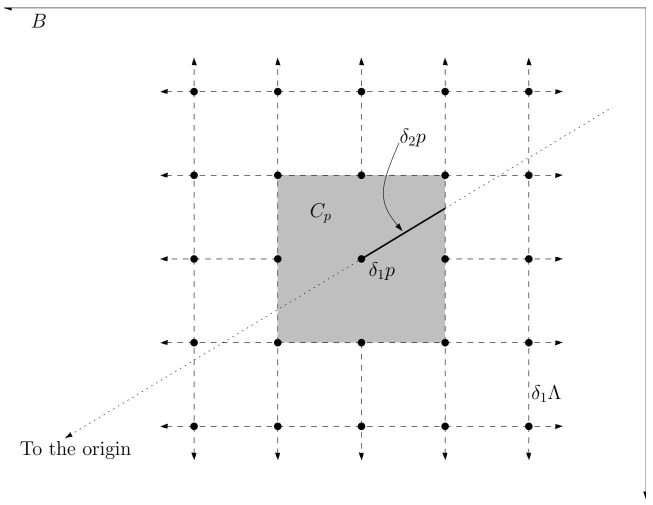


Figure 3.3: If for $p \in \Lambda$ we have $\delta_1 p \in B$, then $\delta_2 p$ lies in one of the cells of $\delta_1\Lambda$ adjacent to $\delta_1 p$.

We are ready for the third part of the proof. Consider the critical values $\delta \in [\delta_1, \delta_2]$ for which the flow of control of the Placement changes. They are the following:

- A point $p \in \Lambda$ such that $\delta p \in B_i$ but $(\delta + \epsilon)p \notin B_i$ or $(\delta - \epsilon)p \notin B_i$ for an infinitesimal $\epsilon > 0$. That is, $\delta p \in \partial B_i$.
- B_i intersects an edge of $\delta\Lambda$, but not of $(\delta+\epsilon)\Lambda$ $(\delta-\epsilon)\Lambda$ for an infinitesimal $\epsilon > 0$.

Because of the property of δ_1, δ_2 stated above, only the vertices V of cells of $\delta_1\Lambda$ that intersect ∂B_i can change the flow of control of Placement. In the L_∞ metric, because the disks are axis-aligned squares, the vertices V are distributed along two axis-aligned rectangles R_1 and R_2. All the vertices of V along the same side of R_1 or R_2 come in or out of B_i at the same time, that is, they intersect ∂B_i for the same value δ. Therefore, each disk B_i induces $O(1)$ such critical values Δ_i changing the flow of control of Placement, and we can compute them in $O(1)$ time.

We can compute all the critical values $\Delta = \bigcup_{i=1}^m \Delta_i$ and sort them in $O(m \log m)$ time. Using a binary search on Δ, we find $\delta_3, \delta_4 \in \Delta$, with $\delta_3 < \delta_4$, such that Placement(δ_3) succeeds but Placement(δ_4) fails. Because $|\Delta| = O(m)$, this can be done in $O(m\sqrt{m}\log^2 m)$ time with $O(\log m)$ calls to Placement. The flow of control of Placement(δ_4) and of Placement($\delta_3+\epsilon$) are the same. Therefore, Placement($\delta_3 + \epsilon$) also fails, and we conclude that Placement(δ_3) yields a 2-approximation because of Observation 3.5. □

Theorem 3.9 *Let $\mathcal{B} = \{B_1, \dots, B_n\}$ be a collection of disks in the plane with the L_∞ metric. We can compute in $O(n\sqrt{n}\log^2 n)$ time a placement $p_1, \dots, p_n$ with $p_i \in B_i$ that yields a 2-approximation of $D(\mathcal{B})$.*

Proof: Let us assume that $r_1 \leq \cdots \leq r_n$, that is, B_i is smaller than B_{i+1}. Consider the values $\Delta = \{\frac{r_1}{3\sqrt{n}}, \dots, \frac{r_n}{3\sqrt{n}}, 4r_n\}$. We know that PLACEMENT$(\frac{r_1}{3\sqrt{n}})$ succeeds, and we can assume that PLACEMENT$(4r_n)$ fails; if it would succeed, then the disks in $\mathcal{B}$ would be disjoint, and placing each point $p_i := c_i$ would give a 2-approximation.

We use PLACEMENT to make a binary search on the values Δ and find a value r_{max} such that PLACEMENT$(\frac{r_{max}}{3\sqrt{n}})$ succeeds but PLACEMENT$(\frac{r_{max+1}}{3\sqrt{n}})$ fails. This takes $O(n\sqrt{n}\log^2 n)$ time, and two cases arise:

- If PLACEMENT$(4r_{max})$ succeeds, then $r_{max} \neq r_n$. In the case that $4r_{max} > \frac{r_{max+1}}{3\sqrt{n}}$, we have a 2-approximation due to Observation 3.4. In the case that $4r_{max} \leq \frac{r_{max+1}}{3\sqrt{n}}$, consider any value $\delta \in [4r_{max}, \frac{r_{max+1}}{3\sqrt{n}}]$. On the one hand, the balls $B_{max+1}, \dots, B_n$ are not problematic because they have degree n in G_δ. On the other hand, the balls $B_1, \dots, B_{max}$ have to be disjoint because $\delta^* \geq 4r_{max}$, and they determine the closest pair in PLACEMENT(δ). In this case, placing the points $p_1, \dots, p_{max}$ at the centers of their corresponding disks, computing the distance $\tilde{\delta}$ of their closest pair, and using PLACEMENT$(\tilde{\delta})$ for the disks $B_{max+1}, \dots, B_n$ provides a 2-approximation.

- If PLACEMENT$(4r_{max})$ fails, then we know that for any $\delta \in [\frac{r_{max}}{3\sqrt{n}}, 4r_{max}]$ the disks B_j with $\frac{r_j}{3\sqrt{n}} \geq 4r_{max}$ have degree at least n in G_δ. We shrink them to have radius $12r_{max}\sqrt{n}$, and then they keep having degree at least n in G_δ, so they are not problematic for the matching. We also use $\mathcal{B}$ for the new instance (with shrunken disks), and we can assume that all the disks have radius $O(12r_{max}\sqrt{n}) = O(r_{max}\sqrt{n})$.

  We group the disks $\mathcal{B}$ into clusters $\mathcal{B}_1, \dots, \mathcal{B}_t$ as follows: a *cluster* is a connected component of the intersection graph of the disks $B(c_1, r_1 + 4r_{max}), \dots, B(c_n, r_n + 4r_{max})$. This implies that the distance between different clusters is at least $4r_{max}$, and that each cluster $\mathcal{B}_j$ can be enclosed in a disk of radius $O(r_{max}|\mathcal{B}_j|\sqrt{n})$.

  For each subinstance $\mathcal{B}_j$, we use Lemma 3.8, where $m = |\mathcal{B}_j|$ and $k = n$, and compute in $O(|\mathcal{B}_j|\sqrt{|\mathcal{B}_j|}\log^2(|\mathcal{B}_j|n))$ time a placement yielding a 2-approximation of $D(\mathcal{B}_j)$. Joining all the placements we get a 2-approximation of $D(\mathcal{B})$, and because $n = \sum_{i=1}^{t} |\mathcal{B}_i|$, we have used

  $$\sum_{j=1}^{t} O\Big(|\mathcal{B}_j|\sqrt{|\mathcal{B}_j|}\log^2(|\mathcal{B}_j|n)\Big) = O(n\sqrt{n}\log^2 n)$$

  time for this last step.

□

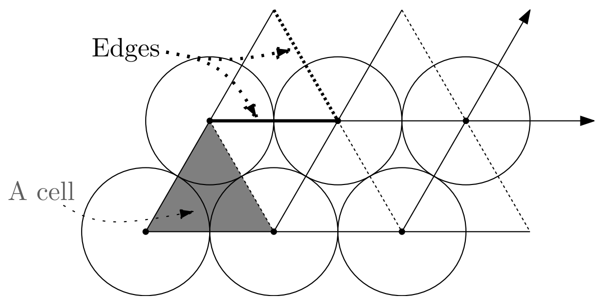


Figure 3.4: Hexagonal packing induced by the lattice $\delta\Lambda = \{\delta(a+\frac{b}{2}, \frac{b\sqrt{3}}{2}) \mid a,b \in \mathbb{Z}\}$. A cell and a couple of edges of $\delta\Lambda$ are also indicated.

3.4 Approximation algorithms in the L_2 metric

We will now study how the L_2 metric changes the bounds and results of the algorithms studied for the L_∞ metric. First, we consider arbitrary disks. Then, we concentrate on congruent disks, for which we can improve the approximation ratio.

3.4.1 Arbitrary disks

For the L_∞ metric, we used the optimal packing of disks that is provided by an orthogonal grid. For the Euclidean L_2 metric we will consider the regular hexagonal packing of disks; see Figure 3.4. For this section, we let

$$\Lambda := \{(a + \frac{b}{2}, \frac{b\sqrt{3}}{2}) \mid a, b \in \mathbb{Z}\}.$$

Like in previous sections, we use $\delta\Lambda = \{\delta p \mid p \in \Lambda\}$. For disks of radius $\delta/2$, the hexagonal packing is provided by placing the disks centered at $\delta\Lambda$. The *edges* of $\delta\Lambda$ are the segments connecting each pair of points in $\delta\Lambda$ at distance δ. They decompose the plane into equilateral triangles of side length δ, which are the *cells* of $\delta\Lambda$; see Figure 3.4.

Consider a version of PLACEMENT using the new lattice $\delta\Lambda$ and modifying it slightly for the cases when B_i contains no lattice point:

- If B_i is contained in a cell C, place $p_i := c_i$ and add the vertices of C to Q; see Figure 3.5a.

- If B_i intersects some edges of $\delta\Lambda$, let E be the edge that is closest to c_i. Then, place p_i at the projection of c_i onto E, and add the vertices of E to Q; see Figure 3.5b.

Observe that, in this case, the distance between a point placed on an edge and a point in $\delta\Lambda \setminus Q$ may be $\frac{\delta\sqrt{3}}{2}$; see Figure 3.5c. We modify accordingly

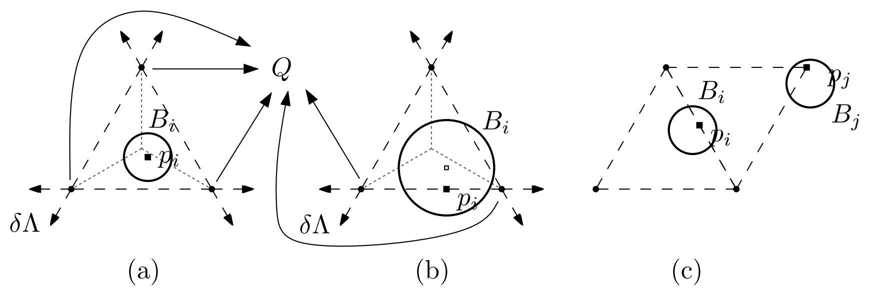


Figure 3.5: Cases and properties of Placement for the L_2 metric. (a) Placement when B_i is fully contained in a cell. (b) Placement when B_i intersects an edge: we project the center c_i onto the closest edge. (c) A case showing that the closest pair in Placement(δ) may be at distance $\frac{\delta\sqrt{3}}{2}$.

the criteria of Definition 3.1 regarding when Placement succeeds, and then we state the result corresponding to Lemma 3.2.

Definition 3.10 *In the L_2 metric, we say that* Placement(δ) succeeds *if the computed placement $p_1, \ldots, p_n$ satisfies $D(p_1, \ldots, p_n) \geq \frac{\delta\sqrt{3}}{2}$. Otherwise,* Placement($\delta$) fails*.*

Lemma 3.11 *If $\frac{4\delta}{\sqrt{3}} \leq \delta^*$, then* Placement($\delta$) *succeeds.*

Proof: We follow the proof of Lemma 3.2. Firstly, we argue that if $\frac{4\delta}{\sqrt{3}} \leq \delta^*$, then G_δ has a matching. Secondly, we will see that if $p_1, \ldots, p_n$ is the placement computed by Placement(δ) when $\frac{4\delta}{\sqrt{3}} \leq \delta^*$, then indeed $D(p_1, \ldots, p_n) \geq \frac{\delta\sqrt{3}}{2}$, that is, Placement($\delta$) succeeds.

Consider an optimal feasible placement $p_1^*, \ldots, p_n^*$ achieving δ^*. We then know that the interiors of $B(p_1^*, \delta^*/2), \ldots, B(p_n^*, \delta^*/2)$ are disjoint. To show that G_δ has a matching, we have to argue that:

- If $B_i \cap \delta\Lambda = \emptyset$, then the points that B_i contributed to Q are in the interior of $B(p_i^*, \delta^*/2)$. We consider both cases that may happen. In case that B_i is fully contained in a cell C of $\delta\Lambda$, then $p_i^* \in C$, and so $C \subset B(p_i^*, \delta) \subset B(p_i^*, \frac{2\delta}{\sqrt{3}}) \subset B(p_i^*, \delta^*/2)$, and the vertices of C are in $B(p_i^*, \delta^*/2)$. In case that B_i intersects edges of $\delta\Lambda$ and p_i was placed on E, then E is the closest edge of $\delta\Lambda$ to c_i and $E \subset B(p_i^*, \frac{2\delta}{\sqrt{3}})$, as can be analyzed in the extreme cases depicted in Figures 3.6a and 3.6b.

- If $B_i \cap \delta\Lambda \neq \emptyset$, we have to argue that there is point $p \in B_i \cap (\delta\Lambda \setminus Q)$. If B_i has diameter smaller than $\delta^*/2$, then $B_i \subset B(p_i^*, \delta^*/2)$ and the points in $B_i \cap \delta\Lambda$ are inside $B(p_i^*, \delta^*/2)$, and so not in Q. If B_i has diameter bigger than $\delta^*/2$, then the region $B_i \cap B(p_i^*, \delta^*/2)$ contains a disk B' of diameter at least $\frac{\delta^*}{2} \geq \frac{2\delta}{\sqrt{3}}$. It is not difficult to see that then B' contains

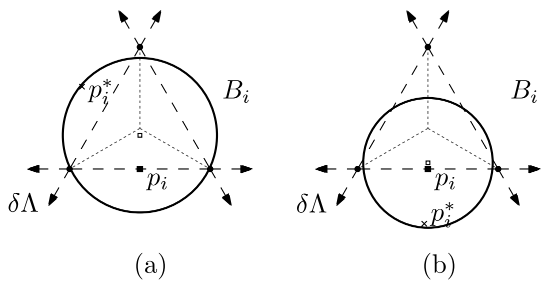


Figure 3.6: Part of the analysis of Placement for the L_2 metric. (a) and (b) When $B_i \cap \delta\Lambda = \emptyset$ and p_i was placed on the edge E, then the distance from p_i^* to any point of E is at most $\frac{\delta\sqrt{3}}{2}$, and therefore $E \subset B(p_i^*, \frac{\delta\sqrt{3}}{2})$.

a point $p \in \delta\Lambda$ (actually this also follows from Lemma 3.16), and so there is a point $p \in \delta\Lambda \cap B' \subset (B(p_i^*, \delta^*/2) \cap \delta\Lambda)$ which cannot be in Q.

This finishes the first part of the proof. For the second part, consider a pair of points p_i, p_j of the placement computed by Placement(δ) when $\frac{4\delta}{\sqrt{3}} \leq \delta^*$. If they have been assigned in the matching of G_δ, then they are distinct points of $\delta\Lambda$ and so they are at distance at least δ. If p_j was assigned in the matching and B_i contains no point from $\delta\Lambda$, then the points in $\delta\Lambda_d \setminus Q$ are at distance at least $\frac{\delta\sqrt{3}}{2}$ from p_i, and so are the distance between p_i and p_j.

If both B_i, B_j do not contain any lattice point, then we know that $r_i, r_j < \frac{\delta}{\sqrt{3}}$, $d(p_i, c_i) \leq \frac{\delta}{2\sqrt{3}}$, and $d(c_i, c_j) \geq d(p_i^*, p_j^*) - d(p_i^*, c_i) - d(p_j^*, c_j) > \frac{4\delta}{\sqrt{3}} - 2\frac{\delta}{\sqrt{3}} = \frac{2\delta}{\sqrt{3}}$. We have the following cases:

- B_i and B_j do not intersect any edge of $\delta\Lambda$. Then $d(p_i, p_j) = d(c_i, c_j) > \frac{2\delta}{\sqrt{3}} > \frac{\delta\sqrt{3}}{2}$.
- B_i intersects an edge E of $\delta\Lambda$, but B_j does not. Then $d(p_i, p_j) \geq d(c_i, c_j) - d(p_i, c_i) - d(p_j, c_j) > \frac{2\delta}{\sqrt{3}} - \frac{\delta}{2\sqrt{3}} - 0 = \frac{\delta\sqrt{3}}{2}$.
- Both B_i, B_j intersect edges of $\delta\Lambda$. See Figure 3.7a to follow the analysis. Without loss of generality, let's assume that c_i lies in the shaded triangle, and so $p_i \in E$, where E is the closest edge to c_i. The problematic cases are when p_j is placed at the edges E_1, E_2, E_3, E_4, E_5, as the other edges are either symmetric to one of these, or further than $\frac{\delta\sqrt{3}}{2}$ from $p_i \in E$. We then have the following subcases:

E_1, E_2. Consider the possible positions of c_j that would induce $p_j \in E_1, E_2$; see Figure 3.7a. The center c_j needs to lie in one of the dotted triangles that are adjacent to E_1 and E_2. But the distance between any point of the dotted triangles and any point of the grey triangle is at most $\frac{2\delta}{\sqrt{3}}$, and so in this case we would have $d(c_i, c_j) \leq \frac{2\delta}{\sqrt{3}}$, which is not possible.

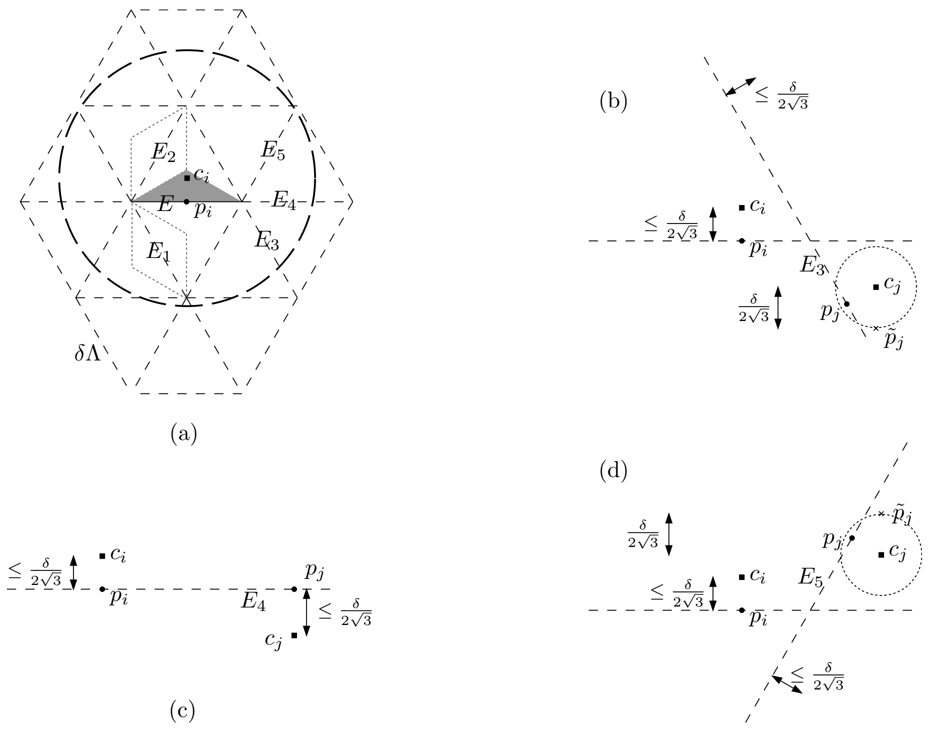


Figure 3.7: Analysis to show that $d(p_i, p_j) \geq \frac{\delta\sqrt{3}}{2}$ when B_i, B_j do not contain any point from $\delta\Lambda$.

E_3. Consider the possible positions of c_j that would induce $p_j \in E_3$; see Figure 3.7b. For a fixed value $d(c_i, c_j)$, the distance between p_i and p_j is minimized when c_i and c_j are on the same side of the line through p_i and p_j, like in the figure. Consider the point $\tilde{p}_j$ vertically below c_j and at distance $\frac{\delta}{2\sqrt{3}}$ from c_j. Then, we have that $d(p_i, \tilde{p}_j) \geq d(c_i, c_j) > \frac{2\delta}{\sqrt{3}}$. Because $d(p_j, \tilde{p}_j) \leq \frac{\delta}{2\sqrt{3}}$, we get $d(p_i, p_j) \geq d(p_i, \tilde{p}_j) - d(p_j, \tilde{p}_j) > \frac{2\delta}{\sqrt{3}} - \frac{\delta}{2\sqrt{3}} = \frac{\delta\sqrt{3}}{2}$.

E_4. Consider the possible positions of c_j that would induce $p_j \in E_4$; see Figure 3.7c. For a fixed value $d(c_i, c_j)$, the distance between p_i and p_j is minimized when c_i and c_j are on opposite sides of the line through p_i and p_j, and $d(p_i, c_i) = d(p_j, c_j) = \frac{\delta}{2\sqrt{3}}$. But, in this case, we can use Pythagoras' theorem to get $d(p_i, p_j) = \sqrt{d(c_i, c_j)^2 - \big(d(p_i, c_i) + d(p_j, c_j)\big)^2} > \sqrt{\left(\frac{2\delta}{\sqrt{3}}\right)^2 - \left(\frac{2\delta}{2\sqrt{3}}\right)^2} = \delta$.

E_5. Consider the possible positions of c_j that would induce $p_j \in E_5$; see Figure 3.7d. For a fixed value $d(c_i, c_j)$, The distance between p_i and p_j is minimized when c_i and c_j are on opposite sides of the line through p_i and p_j, like in the figure. Consider the point $\tilde{p}_j$ vertically above c_j and at distance $\frac{\delta}{2\sqrt{3}}$ from c_j. Then, we have that $d(p_i, \tilde{p}_j) \geq d(c_i, c_j) > \frac{2\delta}{\sqrt{3}}$. Because $d(p_j, \tilde{p}_j) \leq \frac{\delta}{2\sqrt{3}}$, we get $d(p_i, p_j) \geq d(p_i, \tilde{p}_j) - d(p_j, \tilde{p}_j) > \frac{2\delta}{\sqrt{3}} - \frac{\delta}{2\sqrt{3}} = \frac{\delta\sqrt{3}}{2}$.

In all cases, we have $d(p_i, p_j) \geq \frac{\delta\sqrt{3}}{2}$ and this finishes the proof of the lemma. □

Like before, we have the following observations.

Observation 3.12 *If* Placement(δ) *succeeds for* $\mathcal{B}$*, but* Placement(δ) *fails for a translation of* $\mathcal{B}$*, then* $\delta^* \leq \frac{4\delta}{\sqrt{3}}$ *holds and* Placement(δ) *gives an* $\frac{8}{3}$*-approximation.*

If for some $\delta > \delta'$, Placement(δ) *succeeds, but* Placement(δ') *fails, then* $\delta^* \leq \frac{4\delta'}{\sqrt{3}} < \frac{4\delta}{\sqrt{3}}$ *and* Placement(δ) *gives an* $\frac{8}{3}$*-approximation.*

If Placement(δ) *succeeds, but* Placement($\delta + \epsilon$) *fails for an infinitesimal* $\epsilon > 0$*, then* $\delta^* \leq \frac{4\delta}{\sqrt{3}}$ *and* Placement(δ) *gives an* $\frac{8}{3}$*-approximation.*

Lemma 3.6 also applies to the L_2 metric because all the properties of the L_∞ metric that we used in its proof also apply to the L_2 metric.

In the proof of Lemma 3.7 we used a dynamic data structure $\mathcal{D}$ for point sets that supports witness queries: given a disk B_i, report a point contained in B_i. In the L_2 case, we can handle this using a dynamic data structure $\mathcal{D}'$ for nearest neighbor queries: given a point p, report a closest point to p. When we want a witness for B_i, we query with c_i for a closest neighbor p_{c_i}. If p_{c_i} lies in B_i, then we report it as witness, and otherwise there is no point inside B_i.

Using the data structure $\mathcal{D}'$ by Agarwal and Matoušek [5] for the point set P, we can construct the data structure in $O(|P|^{1+\epsilon})$ time, it answers nearest neighbor queries in $O(\log^3 |P|)$ time, and supports updates in $O(|P|^\epsilon)$ amortized time, where $\epsilon > 0$ is an arbitrarily small positive value affecting the constants hidden in the O-notation. In the special case that all the disks are *congruent*, it is better to use the data structure developed by Efrat et al. [62]; it uses $O(|P| \log |P|)$ preprocessing time, it answers a witness query and supports a deletion in $O(\log |P|)$ amortized time. Using these data structures and with the proof of Lemma 3.7, we get the following result for the L_2 metric.

Lemma 3.13 *The Algorithm* Placement *can be adapted to run in* $O(n^{1.5+\epsilon})$ *time. When all the disks are congruent, it can be adapted to run in* $O(n\sqrt{n}\log n)$ *time.*

The running times would actually remain valid for any L_p metric, either using the data structure for nearest neighbors by Agarwal et al. [4] for the general case, or the semi-dynamic data structure of Efrat et al. [62] for congruent disks. However, we would have to use suitable lattices and we would achieve different approximation ratios.

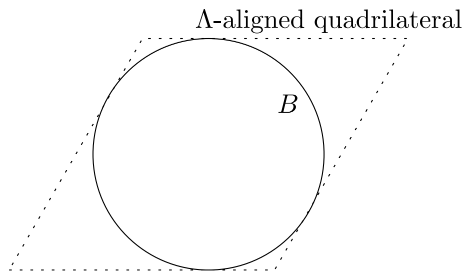


Figure 3.8: For computing δ_1, δ_2 in the proof of Lemma 3.14, we use, instead of B, the smallest Λ-aligned quadrilateral that encloses B.

The proof of Lemma 3.8 is not valid for the L_2 metric because we used the fact that the disks in the L_∞ metric are squares. Instead, we have the following result.

Lemma 3.14 *Let $\mathcal{B}$ be an instance with m disks such that each disk $B_i \in \mathcal{B}$ has radius $O(r\sqrt{k})$, and that there is a disk B of radius $R = O(mr\sqrt{k})$ enclosing all the disks in $\mathcal{B}$. If* Placement$(\frac{r}{3\sqrt{k}})$ *succeeds, then we can compute a placement $p_1, \ldots, p_m$ with $p_i \in B_i$ that yields an $\frac{8}{3}$-approximation of $D(\mathcal{B})$ in $O(mk)$ time plus $O(\log mk)$ calls to* Placement.

Proof: Consider the proof of Lemma 3.8. The first part of it is perfectly valid for the L_2 metric as well.

For the second part of the proof, when computing the values δ_1, δ_2, instead of using the enclosing disk B, we use the smallest Λ-aligned quadrilateral that encloses the disk B; see Figure 3.8. Like in Lemma 3.8, we compute δ_1, δ_2 by making a binary search on the values $\frac{R}{z}$ with $z \in \mathbb{Z} \cap [1, \frac{3R\sqrt{k}}{r}]$. We do not need to compute them explicitly because they are ordered by the inverse of integer numbers. Because $\frac{3R\sqrt{k}}{r} = O(mk)$, we can do this with $O(\log mk)$ calls to Placement.

Like in Lemma 3.8, the values δ_1, δ_2 have the property that if for a point $p \in \Lambda$ there is a $\delta \in [\delta_1, \delta_2]$ such that $\delta p \in \partial B_i$, then ∂B_i must intersect C_p. An easy way to see this is to apply a linear transformation that maps $(1, 0)$ to $(1, 0)$ and $(\frac{1}{2}, \frac{\sqrt{3}}{2})$ to $(0, 1)$. Under this transformation, the lattice $\delta\Lambda$ becomes $\delta\mathbb{Z}^2$, the disk becomes an ellipse, the enclosing Λ-aligned quadrilateral becomes a square enclosing the ellipse, and the proof of the equivalent property follows from the discussion in the proof of Lemma 3.8.

As for the third part of the proof in Lemma 3.8, where we bound the number of critical values Δ_i that a disk B_i induces, we used that in the L_∞ case the disks are squares. This does not apply to the L_2 disks, but instead we have the following analysis.

Because the perimeter of B_i is $O(r_i) = O(r\sqrt{k})$ and we have $\delta_1 = \Omega(r/\sqrt{k})$, the boundary of B_i intersects $O\big(\frac{r\sqrt{k}}{r/\sqrt{k}}\big) = O(k)$ cells of $\delta_1\Lambda$. Together with the property of δ_1, δ_2 stated above, this means that B_i induces $O(k)$ critical values changing the flow of control of Placement. That is, the set $\Delta_i = \{\delta \in [\delta_1, \delta_2] \mid \exists p \in \Lambda \text{ s.t. } \delta p \in \partial B_i\}$ has $O(k)$ values. Each value in Δ_i can be computed in constant time, and therefore $\Delta = \bigcup_{i=1}^{m} \Delta_i$ can be computed in $O(mk)$ time.

Making a binary search on Δ, we find $\delta_3, \delta_4 \in \Delta$, with $\delta_3 < \delta_4$, such that Placement(δ_3) succeeds but Placement(δ_4) fails. If at each step of the binary search we compute the median M of the elements where we are searching, and then use Placement(M), we find δ_3, δ_4 with $O(\log mk)$ calls to Placement plus $O(mk)$ time for computing all medians because at each step we reduce by half the number of elements where to search.

The flow of control of Placement(δ_4) and of Placement($\delta_3 + \epsilon$) are the same. Therefore, Placement($\delta_3 + \epsilon$) also fails, and we conclude that Placement(δ_3) yields an $\frac{8}{3}$-approximation because of Observation 3.12. □

Theorem 3.15 *Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ be a collection of disks in the plane with the L_2 metric. We can compute in $O(n^2)$ time a placement $p_1, \ldots, p_n$ with $p_i \in B_i$ that yields an $\frac{8}{3}$-approximation of $D(\mathcal{B})$.*

Proof: Everything but the time bounds remains valid in the proof of Theorem 3.9. The proof of Theorem 3.9 is applicable. For solving the subinstances $\mathcal{B}_j$ we used Lemma 3.8, and now we need to use Lemma 3.14. Together with Lemma 3.13, it means that for solving the subinstance $\mathcal{B}_j$ we have $m = |\mathcal{B}_j|$ and $k = n$, and so we need to use

$$O(|\mathcal{B}_j|n + |\mathcal{B}_j|^{1.5+\epsilon} \log |\mathcal{B}_j|n)$$

time. Summing over all t subinstances, and because $n = \sum_{j=1}^{t} |\mathcal{B}_j|$, we have spent

$$\sum_{j=1}^{t} O\big(|\mathcal{B}_j|n + |\mathcal{B}_j|^{1.5+\epsilon} \log n\big) = O(n^2)$$

time overall. □

3.4.2 Congruent disks

When the disks $B_1, \ldots, B_n$ are all congruent, say, of diameter one, we can improve the approximation ratio in Theorem 3.15. For general disks, the problematic cases are those balls that do not contain any lattice point. But when all the disks are congruent, it appears that we can rule out those cases. For studying the performance of Placement with congruent disks, we need the following geometric result.

Lemma 3.16 *Let B be a disk of diameter one, and let B' be a disk of diameter $1 \le \delta^* \le 2$ whose center is in B. Consider $\delta = \frac{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-\delta^{*2}}}{4}$. Then,* *the lattice $\delta\Lambda$ has some point in $B \cap B'$. Furthermore, this is the biggest value δ having this property. If B' has diameter $\delta^* \leq 1$, then the lattice $(\delta^*/2)\Lambda$ has some point in $B \cap B'$.*

Proof: Firstly, we consider the case $1 \leq \delta^* \leq 2$ and give a construction showing that $\delta = \frac{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-\delta^{*2}}}{4}$ is indeed the biggest value for which the property holds. Then, we show that $\delta\Lambda$ always has some point in $B \cap B'$ by comparing the different scenarios with the previous construction. Finally, we consider the case $\delta^* \leq 1$.

Assume without loss of generality that the line through the centers of B and B' is vertical. The worst case happens when the center of B' is on the boundary of B. Consider the equilateral triangle T depicted on the left in Figure 3.9. If the center of B is placed at $(0, 0)$, then the lowest point of T is placed at $(1/2 - \delta^*/2, 0)$, and the line L forming an angle of $\pi/3$ with a horizontal line has equation $L \equiv y = 1/2 - \delta^*/2 + \sqrt{3}x$. The intersection of this line with the boundary of B, defined by $y^2 + x^2 = 1/4$, gives the solutions $x = \pm\frac{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-\delta^{*2}}}{8}$. Because of symmetry about the vertical line through the centers of B and B', and because the angle between this line and L, the depicted triangle is equilateral and has side length $\frac{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-\delta^{*2}}}{4}$. This shows that the chosen value of δ is the biggest with the desired property.

We have to show now that the lattice $\delta\Lambda$ has the desired property. It is enough to see that when two vertices of a cell of $\delta\Lambda$ are on the boundary of $B \cap B'$, then the third one is also in $B \cap B'$. In the center of Figure 3.9 we have the case when the two vertices are on the boundary of B. Consider the edge connecting these two points, and its orthogonal bisector. The bisector passes through the center of B, and its intersection with the boundary of B' contained in B is further from it than the intersection of the boundary of B' with the vertical line through the center. Therefore, the third vertex is inside $B \cap B'$.

In the right of Figure 3.9 we have the case where the two vertices are on the boundary of B'. If we consider the case when the lowest edge of the cell is horizontal, we can see that the triangle has the third vertex inside. This is because the biggest equilateral triangle with that shape that is inside $B \cap B'$ has side $\delta^*/2$, and this is always bigger than $\delta = \frac{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-\delta^{*2}}}{4}$ when $1 \leq \delta^* \leq 2$. Then the same argument as in the previous case works.

If one vertex is on the boundary of B and one on the boundary of B', we can rotate the triangle around the first of them until we bring the third vertex on the boundary of B contained in B'. Now we would have two vertices on the boundary of B. If the third vertex was outside $B \cap B'$ before the rotation, then we would have moved the second vertex outside $B \cap B'$, which would contradict the first case. Therefore, the third vertex has to be inside $B \cap B'$.

Regarding the case $\delta^* \leq 1$, we replace the disk B by another disk $\tilde{B}$ of diameter δ^* contained in B and that contains the center of B'. We scale the scenario by $1/\delta^*$ so that both $\tilde{B}$ and B' have diameter 1. If we apply the result we have shown above, we know that $(1/2)\Lambda$ contains some point in $\tilde{B} \cap B'$, and scaling back we get the desired result. $\square$

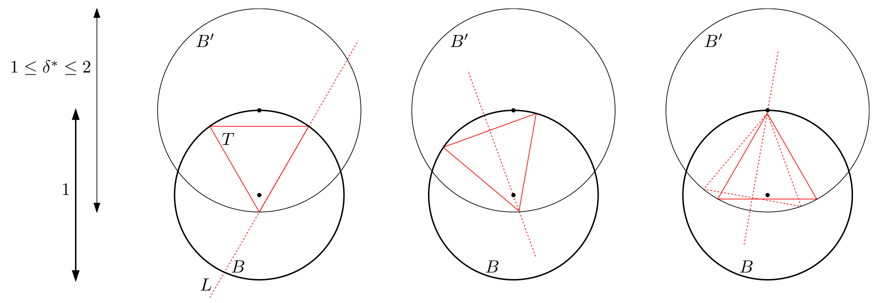


Figure 3.9: Illustration of the proof of Lemma 3.16.

On the one hand, for $1 \leq \delta^* \leq 2$ and $\delta \leq \frac{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-{\delta^*}^2}}{4}$, we have $Q = \emptyset$ when computing Placement(δ), and the graph G_δ has a matching because of Lemma 3.16 and the proof of Lemma 3.11. In this case, if $p_1, \ldots, p_n$ is the placement computed by Placement(δ), we have $D(p_1, \ldots, p_n) \geq \delta$ because all the points $p_i \in \delta\Lambda$. Therefore, for $1 \leq \delta^* \leq 2$, we can get an approximation ratio of

$$\frac{\delta^*}{\delta} \geq \frac{4\delta^*}{-\sqrt{3}+\sqrt{3}\delta^*+\sqrt{3+2\delta^*-{\delta^*}^2}}.$$

For any $\delta^* \leq 1$, the second part of Lemma 3.16 implies that Centers gives a 2-approximation.

On the other hand, we have the trivial approximation algorithm Centers consisting of placing each point $p_i := c_i$, which gives a $\frac{\delta^*}{\delta^*-1}$-approximation when $\delta^* > 1$. In particular, Centers gives a 2-approximation when $\delta^* \geq 2$.

The idea is that the performances of Placement and Centers are reversed for different values δ^* in the interval $[1, 2]$. For example, when $\delta^* = 2$, the algorithm Placement gives a $\frac{4}{\sqrt{3}}$-approximation, while Centers gives a 2-approximation because the disks need to have disjoint interiors to achieve $\delta^* = 2$. But for $\delta^* = 1$, the performances are reversed: Placement gives a 2-approximation, while Centers does not give any constant factor approximation.

The approximation ratios of both algorithms are plotted in Figure 3.10. Applying both algorithms and taking the best of both solutions, we get an approximation ratio that is the minimum of both approximation ratios, which attains a maximum of

$$\alpha := 1 + \frac{13}{\sqrt{65+26\sqrt{3}}} \sim 2.2393.$$

Theorem 3.17 *Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ be a collection of* congruent *disks in the plane with the L_2 metric. We can compute in $O(n^2)$ time a placement $p_1, \ldots, p_n$ with $p_i \in B_i$ that yields a ~ 2.2393-approximation of $D(\mathcal{B})$.*

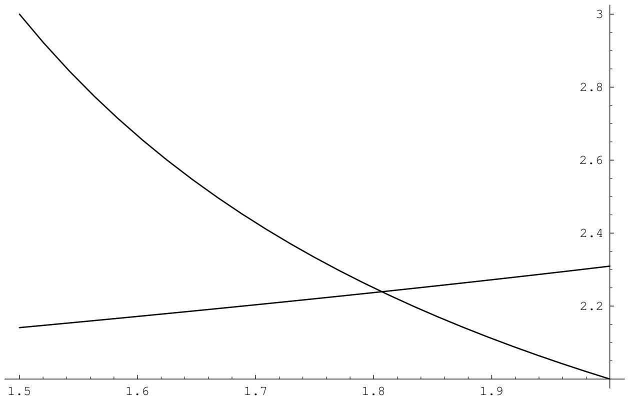


Figure 3.10: Approximation ratios for both approximation algorithms as a function of the optimum δ^*.

Proof: The x-coordinate of the intersection of the two curves plotted in Figure 3.10 is given by

$$\frac{4\delta^*}{-\sqrt{3}+\sqrt{3}\delta^* + \sqrt{3+2\delta^*-{\delta^*}^2}} = \frac{\delta^*}{\delta^*-1}.$$

This solves to $\delta^* := \frac{1}{13}\big(13+\sqrt{13(5+2\sqrt{3})}\big)$, and therefore the approximation ratio is given by $\frac{\delta^*}{\delta^*-1} = \alpha \sim 2.2393$. □

3.5 Concluding remarks

We have presented approximation algorithms for the spreading-points problem. Our approximation ratios rely on packing arguments for the balls in the corresponding metric. However, in the running time of our results, we did not use the regularity of the point sets that we considered. An appropriate use of this property may lead to better running times, perhaps designing data structures for this particular setting.

In the proof of Lemma 3.14, the bottleneck of the computation is that we construct D explicitly. Instead, we could apply randomized binary search. For this to work, we need, for given values δ, δ' and a disk B_i, to take a random point in the set $\tilde{P}_i = \{p \in \Lambda \mid \delta p \in B_i \text{ and } \delta' p \notin B_i, \text{ or vice versa}\}$. For the L_2 metric, we constructed $\bigcup_{i=1}^n \tilde{P}_i$ explicitly in quadratic time, and we do not see how to take a random sample in sub-quadratic time.

The approximate decision problem can be seen as using lattice packings to place disks inside the Minkowski sum $\bigcup_{i=1}^n B_i \oplus B(0, d/2)$. In the L_2 metric, we have used the lattice inducing the hexagonal packing, but we could use a different lattice. For the L_2 metric, the rectangular packing gives a worse approximation ratio, and it seems natural to conjecture that the hexagonal packing provides the best among regular lattices. On the other hand, deciding if better approximation ratios can be achieved using packings that are not induced by regular lattices seems a challenging problem.

Chapter 4

Testing homotopy

In this chapter we present efficient algorithms for a basic topological question, namely, testing if two given paths are homotopic; that is, whether they wind around obstacles in the plane in the same way. Specifically, suppose that the input consists of a set P of up to n points in the plane, and two paths, α and β, that start and end at the same points and are represented as polygonal lines of at most n segments each. The goal is to determine whether α is deformable to β without passing over any points of P; see Figure 4.1. Equivalently, we determine whether the closed loop $\alpha\beta^R$, which concatenates α with the reverse of β, is contractible in the plane minus P. We assume (or simulate) general position, so that no three points are colinear and no two points are on the same vertical line. We are primarily concerned with paths that have no self-intersections.

The path homotopy question arises in several application areas: In circuit board design, *river routing*, where the homotopy class of each wire is specified, is one of the few polynomial-time solvable variations of the wire routing problem [74, 96]. In motion path planning, one may check to see that two ways of getting from point A to point B are equivalent [85]. In Cartography and Geographic Information Systems (GIS), one may wish to simplify a linear feature (road or river) while respecting the way in which the feature winds around points [27, 46]. Michael Goodchild, in an invited lecture at the 11th ACM Symposium on Computational Geometry, pointed out that even on a road map that

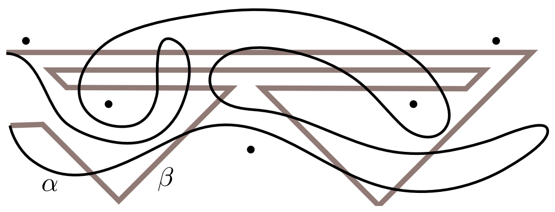


Figure 4.1: Are α and β homotopic?

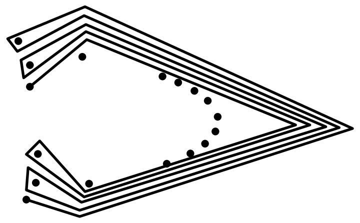


Figure 4.2: $\Theta(n^2)$ segments in the shortest homotopic path

has features with 60m accuracy, you will still find all the houses on the proper side of the road. In such a case, the operators entering data have used topological constraints to make sure that the road winds properly when creating the road layer or building layer. Efrat et al. [64] recently looked at computing shortest paths among obstacles by identifying and bundling homotopic fragments. They independently developed a sweep algorithm that uses ideas similar to ours. Our rectification can be used to speed up their algorithm, which is further improved by Bespamyatnikh [21].

Several approaches to test path homotopy among points in the plane give algorithms with quadratic worst-case behavior. One approach is to find the Euclidean shortest representatives of the homotopy classes for α and β using known algorithms, then to check that they are identical. These algorithms [85] trace each path through a triangulation, taking time proportional to the number of triangles that intersect the paths, which may be $\Theta(n^2)$. In fact, explicit representation of the shortest path may take $\Theta(n^2)$ segments for a path like the one in Figure 4.2. The more general problem of testing if two paths are homotopic in compact surfaces, with or without boundary, was considered by Dey and Guha [49] and Dey and Schipper [50]. Here the specification of the path essentially must be as a sequence of the edges that are crossed in the universal cover of the surface, which may again be quadratic.

For simple paths, our algorithm runs in $O(n \log n)$ time, which we show is tight. The approach is to convert α and β separately to near-canonical representations of their homotopy classes, then compare the representations to determine if the paths are homotopic. For self-intersecting paths, our algorithm runs in $O(n^{3/2} \log n)$ time.

After laying the topological groundwork in the next section, we show how to test homotopy for simple paths in Section 4.2. We then describe our algorithm for non-simple paths in Section 4.3. We establish lower bounds for both problems in Section 4.4, in particular showing that the non-simple case is related to Hopcroft's problem. We conclude in Section 4.5.

4.1 Topological preliminaries

We actually solve three natural variations of the path homotopy question, so clear definitions are important.

4.1.1 Three variations on path homotopy

The topological concept of homotopy formally captures the notion of deforming paths [9, 106]. Let $I = [0, 1]$ denote the unit interval and M denote a topological space, which for us will be the complement of some point or polygonal obstacles in the plane. A *path* is a continuous function $\alpha\colon I \to M$; that is, a function for which the preimage $\alpha^{-1}(A)$ of an open set $A \subseteq M$ is open in I. Paths α and β that share starting and ending points, $\alpha(0) = \beta(0)$ and $\alpha(1) = \beta(1)$, are said to be *path homotopic* if one can be deformed to the other in M while keeping the endpoints fixed. The formal definition is as follows [43, 82].

Definition 4.1 *Two paths α, β with the same endpoints are* equivalent *(or* homotopic*) in M if and only if there exists a continuous function $F : [0, 1]\times[0, 1] \to M$ such that:*

- *$F(0, t) = \alpha(t)$ and $F(1, t) = \beta(t)$, for all $t \in [0, 1]$ (the first path is α, the final path is β),*
- *$F(s, 0) = \alpha(0) = \beta(0)$ and $F(s, 1) = \alpha(1) = \beta(1)$, for all $s \in [0, 1]$ (the endpoints are fixed).*

By the standard topological definition, path endpoints must lie in the space M, and the path can pass over them by continuous deformation, creating self-intersections. This may be undesirable in some applications, so we also consider two alternative definitions that allow a path to begin or end at obstacles in the plane. Informally, we can think of the path as a thread winding above a plane that is punctured with long needles that serve as obstacles. If we fix the ends of the thread with tacks, pins, or pushpins, we can obtain three variations of the homotopy problem, as defined below.

tack A thumbtack pushed flat into the plane presents no obstacle to the thread, so this corresponds to the standard topological definition given above.

pin A straight pin serves as a point obstacle, so that the endpoints p_0 and p_1 are not included in the topological space M. A *path* is a continuous function from an open interval $\alpha\colon (0, 1) \to M$ such that the one-sided limits $\lim_{x\to 0^+} \alpha(x) = p_0$ and $\lim_{x\to 1^-} \alpha(x) = p_1$.

pushpin A pushpin serves as a larger obstacle, so that closed ϵ neighborhoods around the the endpoints are not included in the topological space M. A *path* is defined as in the pin case, but now the limit points p_0 and p_1 are chosen on the boundaries of the ϵ neighborhoods. This is equivalent to fixing the path direction at the endpoint, or to adding point obstacles infinitesimally to the left and to the right of the path endpoint.

The two examples of Figure 4.3 show that these definitions lead to different notions of path homotopy. Paths α and β are homotopic under the tack definition, but not under either the pin or pushpin definitions. Paths γ and δ are homotopic under the tack and pin definitions, but are not homotopic under the pushpin definition, which preserves how γ winds around the left endpoint.

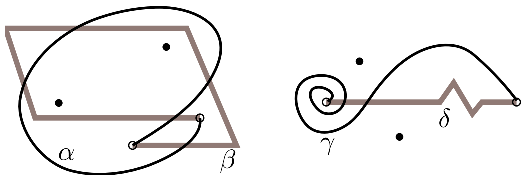


Figure 4.3: Distinguishing between path definitions

Because we can add obstacles to the plane to handle the pin and pushpin definitions, we consider the tack (standard) definition in greatest detail. This does cause extra complications in handling self-intersections, especially in Section 4.2.6. The works of Efrat et al. [64] and Bespamyatnikh [21] use the pin and pushpin definitions.

4.1.2 Canonical sequences

Whatever definition of path we choose, path homotopy is an equivalence relation [9, 106], which implies we can identify a homotopy class by giving a representative path.

We can write a simple representation of a path as the sequence of points that it passes above and below. We can depict this by drawing vertical rays upward and downward from each point of P to form vertical *slabs.* For a point p, we use l_p^+ and l_p^- to denote the vertical halfline with point p as lowest and highest point respectively. The sequence just records the intersections with these rays as we traverse a path through the slabs. Figure 4.4 illustrates the paths with sequences $\alpha \equiv l_{p_1}^- l_{p_2}^- l_{p_2}^- l_{p_2}^+ l_{p_3}^+ l_{p_4}^+ l_{p_4}^- l_{p_3}^+ l_{p_3}^+ l_{p_4}^- l_{p_5}^- l_{p_5}^- l_{p_4}^- l_{p_3}^+ l_{p_2}^- l_{p_1}^-$ and $\beta \equiv l_{p_1}^- l_{p_2}^+ l_{p_3}^+ l_{p_4}^+ l_{p_5}^- l_{p_5}^- l_{p_4}^- l_{p_4}^+ l_{p_4}^+ l_{p_3}^+ l_{p_2}^+ l_{p_2}^+ l_{p_2}^- l_{p_1}^-$. This representation can be constructed for paths with or without self-intersections.

A repeated ray with opposite signs is a *turn point.* Our α sequence has two turn points, $l_{p_2}^- l_{p_2}^+$ and $l_{p_4}^+ l_{p_4}^-$. Polygonal path α can have at most $n-1$ turn points, since there must be at least one vertex of α between intersections with two rays from the turn point, and each vertex is claimed by at most one turn point.

An adjacent pair of repeated symbols can be deleted by deforming the curve out of a slab without changing the homotopy class. This deletion can be repeated until we obtain a *canonical sequence.* (The canonical sequence may even be empty.) For example, from α we delete $l_{p_2}^- l_{p_2}^-$ and $l_{p_4}^- l_{p_3}^+ l_{p_3}^+ l_{p_4}^- l_{p_5}^- l_{p_5}^-$, and from β we delete $l_{p_5}^- l_{p_5}^-$, $l_{p_4}^+ l_{p_4}^+$, and $l_{p_2}^+ l_{p_2}^+$, to find that both have the same canonical sequence $l_{p_1}^- l_{p_2}^+ l_{p_3}^+ l_{p_4}^+ l_{p_4}^- l_{p_3}^+ l_{p_2}^- l_{p_1}^-$. In fact, two paths have the same canonical sequence if and only if they are homotopic. This is most easily seen using universal covers, which we sketch in the next subsection. (Unfortunately, canonical sequences can have quadratic length, so we cannot store and manipulate them explicitly.)

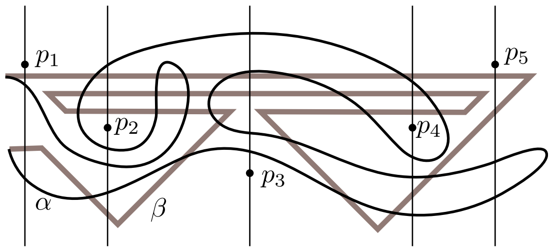


Figure 4.4: Sequences $\alpha \equiv l^-_{p_1} l^-_{p_2} l^-_{p_2} l^+_{p_2} l^+_{p_3} l^+_{p_4} l^-_{p_4} l^+_{p_3} l^+_{p_3} l^-_{p_4} l^-_{p_5} l^-_{p_5} l^-_{p_4} l^+_{p_3} l^-_{p_2} l^-_{p_1}$ and $\beta \equiv l^-_{p_1} l^+_{p_2} l^+_{p_3} l^+_{p_4} l^-_{p_5} l^-_{p_5} l^-_{p_4} l^+_{p_4} l^+_{p_4} l^+_{p_3} l^+_{p_2} l^+_{p_2} l^-_{p_2} l^-_{p_1}$ lead to the same canonical sequence $l^-_{p_1} l^+_{p_2} l^+_{p_3} l^+_{p_4} l^-_{p_4} l^+_{p_3} l^-_{p_2} l^-_{p_1}$.

4.1.3 Covering space

The topological concept of covering space has been used in the work of Gao et al. [74] and Hershberger and Snoeyink [85]. We briefly describe its properties here, and refer to basic topology texts [9, 106] for a more thorough, complete description.

Informally, a topological space U is a covering space of a space X if, at each point $u \in U$, there is a corresponding point $x \in X$ such that things around u and x look the same in their respective spaces, but there may be many points of U mapping to the same point x.

Formally, let $p\colon U \to X$ be a continuous and onto map between connected topological spaces U and X. If every point $x \in X$ has an open neighborhood N where the inverse image $p^{-1}(N)$ is a union $\bigcup_i U_i$ of disjoint open sets of U and the restriction $p|_{U_i}$ is a homeomorphism from U_i onto N, then p is a covering map and U is a covering space of X.

A space is always a covering space of itself under the identity map [9, 106]. A more interesting covering space is the universal covering space, which is simply connected—every loop in this space can be contracted to a point. In our setting (Figure 4.4), this space is most easily described by a procedure that grows a region by gluing together copies of vertical slabs at their boundary rays. Start with a region that consists of any single vertical slab, and therefore has four boundary rays (or two if we started with the leftmost or rightmost slab.) Then loop forever, selecting a boundary ray and gluing on a copy of the missing slab along that ray. Never form a cycle or enclose a ray's endpoint.

When the set of obstacle points, P, is non-empty, the universal covering space is infinite, which is why our procedure does not terminate. It is relatively easy to construct only the portions of the universal cover that intersect the given paths α and β because any path can be *lifted* to the universal cover by choosing the sequence of rays and slabs in the order that a path intersects them. In fact, every path in the plane minus P has a unique lift into the universal cover once the starting point is specified [9, 106]. To test path homotopy, one can simply lift both α and β to the universal cover starting from the same point and ask if they end at the same point in the universal cover.

Because the universal cover is simply connected, the dual graph, with a vertex for each slab, is an infinite tree. The dual of the portion visited by a path is a finite tree, and the operation of constructing a canonical sequence prunes leaves from this tree so that what remains is the unique shortest path from the slab of the start point to the slab of the endpoint. This establishes the difficult-to-prove direction of the following lemma.

Lemma 4.2 *Two paths have the same canonical sequence if and only if they are homotopic.*

Proof: It is clear that two paths with the same canonical sequence are homotopic, since the construction of a canonical sequence from a path is a concatenation of homotopies on slabs. For the reverse, suppose that paths α and β are homotopic. When we lift both to the universal cover, starting at the same point, they end at the same point. Their canonical sequences, therefore, must go from the same starting slab to the same ending slab. There is only one shortest path that does so in the tree, so the canonical sequences are the same. □

The following corollary will be useful in proving that paths have an empty canonical sequence.

Corollary 4.3 *Any path that can be lifted to the universal cover to start and end on the same vertical segment has an empty canonical sequence.*

Proof: The path is homotopic to the vertical segment, which has an empty canonical sequence. □

4.2 Homotopy test for simple paths

In this section we focus on homotopy testing for paths α and β that are *simple*—they have no self-intersections. An aboveness ordering allows us to find a rectified representation of a path. Using orthogonal range queries, we can adjust the path to have the canonical sequence, then test homotopy for two adjusted paths in $O(n \log n)$ time.

4.2.1 Aboveness ordering

For sets in the plane one can define an *aboveness relation* that is useful in many algorithms in computational geometry [110]: $A \succ B$ if there are points $(x, y_A) \in A$ and $(x, y_B) \in B$ with $y_A > y_B$.

We say that a set is *vertically convex* if it is path connected [9, 106] and the intersection with any vertical line is an (open or closed) interval. Break a path α into x-monotone fragments and conceptually separate them at their common endpoints to give a collection of disjoint vertically convex sets. When applied to disjoint, vertically convex sets, we can easily show that the aboveness relation is a partial order [18]:

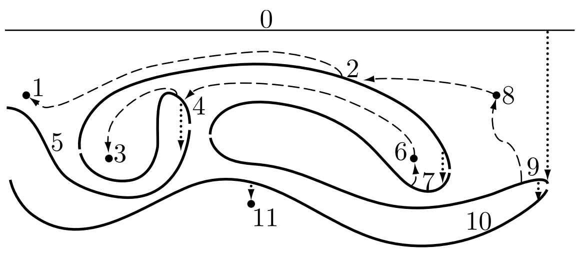


Figure 4.5: The aboveness tree for path α, and its inorder numbering in the child/sibling representation.

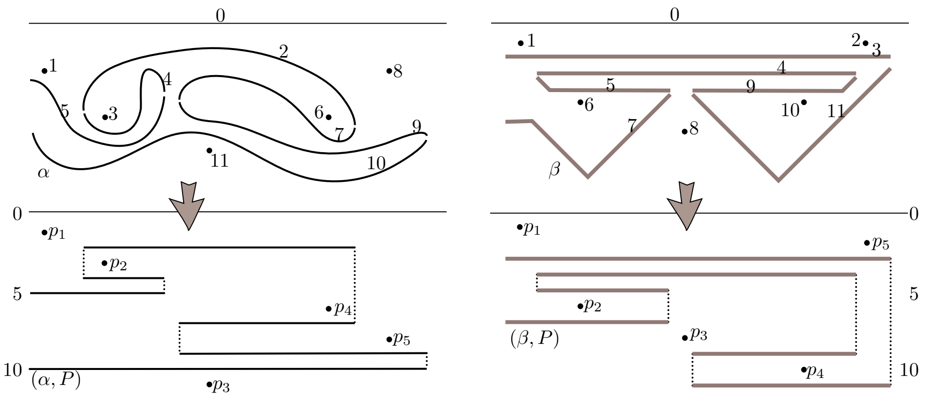


Figure 4.6: Numberings for α and β are used to form rectified pairs (α, P) and (β, P). Rectifying preserves path simplicity and sequences.

Lemma 4.4 *The aboveness relation is acyclic for pairwise disjoint, vertically convex sets $A_1, \ldots, A_k$ in the plane.*

It is easy to develop a plane sweep algorithm that can compute in $O(n \log n)$ time, a total order for monotone fragments of a path and points that is consistent with the aboveness relation. We sketch an algorithm adapted from Palazzi and Snoeyink [109].

Define an *aboveness tree* on disjoint points and monotone path fragments, in which each point and path is a child of the path directly above its rightmost endpoint. This is a k-ary tree, in which children can be ordered left to right by rightmost endpoints. We add a horizontal segment at $y = \infty$ to serve as the root of this tree. For disjoint paths specified by n segments, this tree can be constructed in $O(n \log n)$ time by a plane sweep algorithm.

We represent this k-ary tree as a binary tree by giving each path a pointer to its left sibling and rightmost child, as in Figure 4.5. We then number the points and paths according to an inorder traversal that recursively visits the left sibling, the node, and then the rightmost child.

We can prove that this numbering is consistent with the aboveness relation. Define a *rightward trace* from any point or monotone path by tracing paths to their rightmost endpoints, and to parent pointers, up to the root. Note that traces do not cross. If $s \succ t$ then a trace from s must come from the left to meet a trace from t, and the inorder traversal numbers the subtree containing s before the subtree containing t. See [109] if more detail is desired.

4.2.2 Rectified pairs

We can use a total ordering that respects the aboveness to rectify any simple path α around points P. Simply rank each point of P and monotone fragment of the path and replace all y coordinates by their ranks. We call the result a *rectified pair* (α, P).

Figure 4.6 shows the rectified pairs (α, P) and (β, P). Each path is considered to consist of horizontal segments that come from monotone path fragments and vertical segments that connect these fragments in order.

Notice that the points receive different ranks for the numberings of the two paths, and therefore different y coordinates in the two rectified pairs. The points can still be distinguished by their x coordinates, which were assumed to be distinct and do not change. All aboveness relationships are preserved, so rectifying a pair does not change a path sequence or cause self-intersection.

4.2.3 Orthogonal range queries

Rectifying a pair makes it easier to compute a canonical sequence, because we can search for turn points using orthogonal range queries. Our problem is to preprocess a set of points P in the plane to answer queries of the form "Report the rightmost point in an axis-aligned query rectangle Q, or 'none' if the rectangle Q is empty." (We use a symmetric structure to query for leftmost points.) These are sometimes known as three-sided range queries, because it is sufficient to supply the top, bottom, and right side of the query rectangle. This problem can be solved in $O(n \log n)$ preprocessing time and linear space using Chazelle's data structure for segment dragging [38]. The RT of Edelsbrunner [57] or the range priority tree of Samet [113] achieve the same time bound, with an extra logarithmic factor in space but a savings in programming complexity.

Lemma 4.5 *Three-sided range queries can be answered in $O(\log n)$ time, with $O(n \log n)$ preprocessing and linear space.*

4.2.4 A rectified canonical path

Given a rectified pair (α, P), we are going to describe an algorithm Rcp (shorthand for Rectified_canonical_path) that computes a new path whose sequence is the canonical sequence for α. We also call this new path a *rectified canonical path*. It must be understood that this is defined with respect to a rectified pair (α, P), and not the original path and points. At the end of this

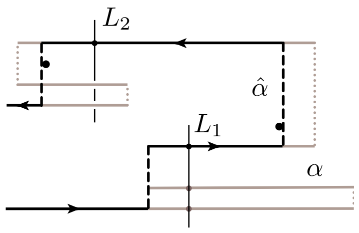


Figure 4.7: The path $\hat{\alpha}$ has the last intersection property.

subsection, we derive non-rectified canonical paths for α, β, and the original point set P.

The algorithm RCP is a simple, incremental procedure to compute a rectified canonical path from the pair (α, P). We assume that α is represented by the list x_0, y_0, x_1, y_1, $\ldots$, y_{n-1}, x_n, which indicates that the path visits (x_0, y_0), (x_1, y_0), (x_1, y_1), $\ldots$, (x_{n-1}, y_{n-1}), (x_n, y_{n-1}), alternating between horizontal and vertical segments. We think of α as oriented from the start to the end.

The canonical path that we compute is determined from (α, P) by an important *last intersection property.* Given any oriented curve γ in the plane, we say that a curve $\hat{\gamma}$ from the homotopy of γ has the last intersection property if any vertical segment L in the universal cover intersecting the lift of $\hat{\gamma}$ does so at the last of its intersections with the lift of γ. In Figure 4.7, the dark curve $\hat{\alpha}$ has the last intersection property in the universal cover. For example, segment L_1 has three intersections with the lift of α, and the last is in the lift of $\hat{\alpha}$. Segment L_2 actually has only one intersection with the lift of α, which is in $\hat{\alpha}$.

We consider the horizontal segments of α in order. We maintain the invariants that after i segments, for $0 \leq i \leq n$,

- stack S contains a rectified canonical path for the prefix of α from x_0 to x_i,
- the canonical sequence for the path on S has no adjacent repeated symbols, and
- the path on S has the last intersection property.

Begin by setting $i = 0$ and pushing x_0 onto S. This establishes the initial invariant for the stack; the others are trivially true.

While $i < n$, let X denote the top element on the stack, and if $X \neq x_0$, let Y_s and X_s be the pair just below X on the stack. Figure 4.8 illustrates cases (I)–(V) as we consider the current segment from (X, y_i) to (x_{i+1}, y_i). We assume that $x_{i+1} < X$, since the reverse is symmetric.

If we are not doubling back—that is, if (I): $X = x_0$ or (II): $x_{i+1} < X < X_s$—then push y_i and x_{i+1} onto S, and increment i. The canonical sequence for this path will not gain any repeated pair of symbols and the path is extended in the current direction.

If we are doubling back, $X_s \leq X$ and there are three cases. Either (III): there is a turn point p with $\max\{X_s, x_{i+1}\} < p.x \leq X$, or there is no turn point

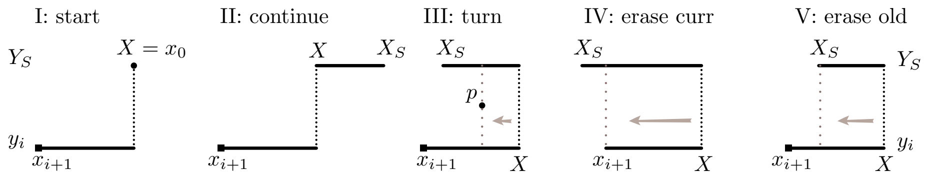


Figure 4.8: Five cases in incrementally computing a canonical path.

and (IV): $X_s < x_{i+1} < X$ or (V): $x_{i+1} < X_s \leq X$. We check for a turn point by performing an orthogonal range query with the rectangle $[\max\{X_s, x_{i+1}\}, X] \times [Y_s, y_i]$. We handle each case separately.

(III): Turn point $(p.x, p.y)$ is found, so replace X with $p.x$ at the top of S, then push y_i and x_{i+1} onto S, and increment i. Because of the turn point, adjacent symbols in the canonical sequence will not be identical, and vertical lines in the universal cover will intersect a single horizontal segment. (Note: we may wind over and under p in this case; sometimes it helps to think of p as a pair of points separated infinitesimally in x so that winding generates infinitesimal horizontal segments.)

(IV): No turn point; erase current segment. Replace X with x_{i+1} at the top of S and increment i. This shortens the path stored in S, so cannot create repeated symbols or violate the last intersection property.

(V): No turn point; erase old segment. Pop stack S twice to remove the old segment at the top. This case may apply repeatedly, as it does not increment i, but each repetition shortens the path in S.

We continue until $i = n$, and use Rcp(α, P) to denote the output.

Theorem 4.6 *The algorithm* Rcp *correctly computes a path whose sequence is the canonical sequence for* (α, P) *using at most* $2n$ *orthogonal range queries plus* $O(n)$ *time.*

Proof: First, we show that the algorithm terminates after $2n$ iterations. Each of the cases (I)–(IV) increments the loop variable i and pushes at most two elements onto S. Case (V) pops S twice and leaves i unchanged. Initially $i = 0$, and we stop when $i = n$, so we have n iterations in total with cases (I)–(IV) and at most $2n$ elements pushed onto S. Case (V) never empties the stack—the sentinel x_0 triggers case (I)—so we have at most n iterations with case (V).

Using the arguments given in the description of the cases, we can establish and inductively maintain the invariants that, after i segments, stack S contains a rectified canonical path, with the last intersection property, that starts from x_0 and ends with x_i. When the algorithm terminates at $i = n$, we have the rectified canonical path. □

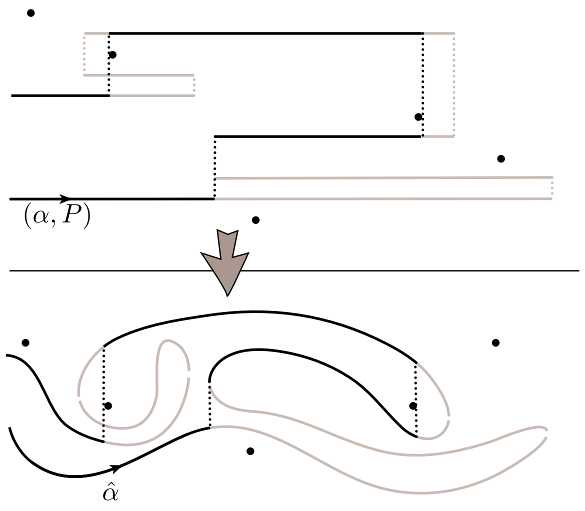


Figure 4.9: Unrectifying the canonical path $\text{RCP}(\alpha, P)$

Once we have run the algorithm, we can un-rectify the path, as in Figure 4.9, to return to the original set of points. This will allow us to compare two paths on the same point obstacles. The rectified canonical path uses horizontal segments, corresponding to segments of α, and vertical segments corresponding to portions that have been shrunk by homotopy. Since the x-coordinates of all points and segments were preserved in rectification, and the aboveness relation was respected, we can easily map back to the original portions of α and join them by vertical segments, forming canonical path $\hat{\alpha}$. The last intersection property (recall Figure 4.7) is also preserved.

We have described the algorithm for the standard (tack) definition of path homotopy. For the pin and pushpin definitions, we add the two path endpoints to the set of obstacle points P. Under the pushpin definition, we also add two points that are infinitesimally left and right of the start point to make the algorithm wind correctly around the start. Under the pin definition, the algorithm unwinds at the start, and we postprocess, if needed, to unwind at the endpoint.

4.2.5 A leftist path

Figure 4.10(a), adapted from Efrat et al. [64], illustrates that a canonical path with the last intersection property need not be simple. In fact, one can create examples with a quadratic number of self-intersections. We define the *leftist canonical path* $\hat{\alpha}$ and prove that under the pin and pushpin definitions it has no self-intersections and under the tack definition it has $O(n)$ intersections.

The path is called "leftist" because it consists of monotone fragments that are traversed from left to right, so the vertical segments chosen for homotopy

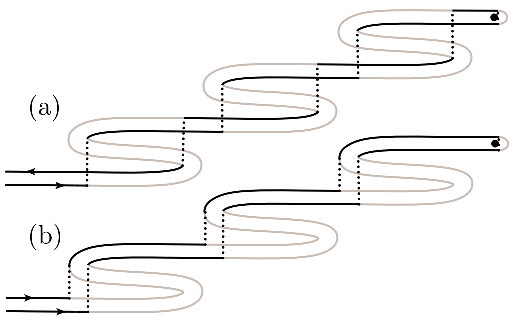


Figure 4.10: Self-intersections in (a) avoided by leftist path (b).

are as far to the left as possible, as in Figure 4.10(b). The idea is simple: put together fragments of two canonical paths to change the last intersection property to a *left intersection property:* any vertical segment in the universal cover either does not intersect the lift of the path, or intersects it in the point where the lift of α last crosses from left to right.

For a rectified pair (α, P), let α^F and α^R (forward and reverse) be the two canonical paths obtained by running the algorithm of the previous section twice, starting from either end of the curve α. Since both paths are homotopic to α, the canonical sequence of α^R is the reverse of that of α^F—both have the same turn points, which we can number p_1 to p_{m-1} along α^F. (Some points may be duplicated, and we use infinitesimal horizontal segments for portions that wind around a point.) Let p_0 denote the start point and p_m the endpoint.

We snip α^F and α^R at the turn points. Fragments ending below a turn point are extended infinitesimally beyond the turn point by an amount proportional to their rank order below the turn point. For example, if there are three fragments whose right endpoints are below a turnpoint $p \in P$, then the lowest one is extended 3ε to the right, the next 2ε and the highest by ε, for an infinitesimal $\varepsilon > 0$. To assemble these fragments into the *leftist path* $\hat{\alpha}$, we start at point p_0, with $i = 1$. We take the fragment α_i from α^F if p_{i-1} is left of p_i, otherwise we take the fragment α_i from α^R. We connect fragments in order, extending fragments above turn points to match the extensions below.

Notice that $\hat{\alpha}$ has the same canonical sequence as α—since it was derived from fragments of α^F and α^R, it has the same turn points and the same sets of points above and below. Thus, $\hat{\alpha}$ is homotopic to α.

A vertical segment within a fragment $\alpha_i \subset \hat{\alpha}$ has the left intersection property, because it had the last intersection property with either α^F or α^R, whichever came from the left. At the left endpoints, fragments have the left intersection property for any vertical line just to the right of the obstacle point; at the right endpoints, they have the left intersection property for vertical lines just to the left of the obstacle point. We use these properties as we consider self-intersections of the leftist path $\hat{\alpha}$ in the next lemma.

Lemma 4.7 *Consider the leftist canonical path $\hat{\alpha}$ for a simple path α with n*

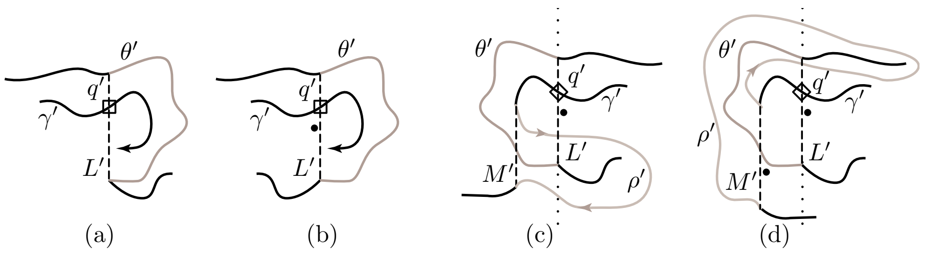


Figure 4.11: Examples lifting γ' starting from q' within $R' = R(L', \theta')$. (Striped areas are used toward the end of the proof of Lemma 4.7.)

segments: Under the pin and pushpin definitions $\hat{\alpha}$ is also simple. Under the tack definition $\hat{\alpha}$ has at most $2n$ points of self-intersection.

Proof: On the leftist path $\hat{\alpha}$, consider any vertical segment $L \subset \hat{\alpha}$, from the homotopy of a portion $\theta \subset \alpha$. Let p be the start point of curve α, and let γ be the connected portion of $\alpha \setminus \theta$ that contains p. We lift γ to the universal cover to consider whether a point q where γ intersects L can be a self-intersection of $\hat{\alpha}$.

Lift L and θ from either of their shared endpoints to obtain L' and θ', which bound a simply-connected region $R' = R(L', \theta')$ in the universal cover. Lift intersection point q to $q' \in L'$, and lift curve γ to γ' starting from point q'. If this lifts p to a point p' that lies in R', then we say that q is a *normal* intersection, otherwise q is *special*. Point q is normal in Figure 4.11(a–b) and special in (c–d). We aim to count self-intersections of $\hat{\alpha}$ by separately counting the normal and special intersections that can create them. Figure 4.12 shows a complex example (under the tack definition) in which $L \cap \gamma$ has both normal and special intersections that appear as self-intersections of $\hat{\alpha}$.

Normal intersections actually come in equivalence classes: a given lift of p to $p' \in R'$ can contribute several intersections $L' \cap \gamma'$, all of which will be normal. We consider all of these to be in the same equivalence class, by Corollary 4.3. In fact, each class can contribute at most one self-intersection of $\hat{\alpha}$ by the left intersection property. We therefore count the number of normal intersection classes.

Under the pin and pushpin definition, there are no normal intersection classes—endpoint p is an obstacle, so p cannot be lifted to lie inside the simply connected region R'.

Under the tack definition, the number of normal intersection classes for L and γ is the number of copies of point p that can be lifted into R'. (Two in Figure 4.12, for example.) To count copies, from each copy p' in the universal cover draw a vertical ray downward to hit a segment of θ'; each segment of θ' (except the one hitting the top of L') can be hit by at most one ray in the universal cover, since all the vertical rays have the same projection into the plane. Thus, R' contains at most $|\theta|$ distinct copies of p. The total number of normal classes over all vertical segments of $\hat{\alpha}$ is $2n$, since each vertical segment

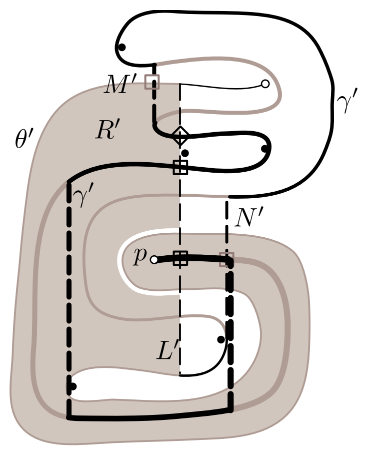


Figure 4.12: $L \cap \gamma$ has normal intersections in two classes and special intersections.

contributes a unique contracted curve and at most two remaining fragments of α.

It remains to count special points that are self-intersections of $\hat{\alpha}$. In the remainder of the proof, we show that each special self-intersection q can be mapped one-to-one to a normal class that contributes no normal self-intersection to $\hat{\alpha}$. (Thus the number of normal and special self-intersections is bounded by the number of normal classes.) The one special self-intersection in Figure 4.12 (marked with a diamond) maps to the normal class of $M' \cap \theta'$.

Therefore, consider a special point $q \in L \cap \gamma$ that is a self-intersection of $\hat{\alpha}$. The lift γ' enters R' by crossing L' at q', but by the definition of special, γ' cannot end inside R'. Since γ' does not cross θ', it must leave by crossing L' again. If R' was to the right of L' at q', as in Figure 4.11(a–b), this would violate the left intersection property, so we know that R' lies to the left of L'.

The lift of $\hat{\alpha}$ must exit R' by a vertical segment M' to the left of L', as in Figure 4.11(c–d). This M' has its own region $R(M', \rho')$, for some $\rho \subset \gamma$. Since M' exits R' without crossing L', the curve θ' must cross M' an odd number of times, and therefore θ' ends inside $R(M', \rho')$.

Let ℓ denote the line through L' in the universal cover. Since θ' ends on ℓ inside $R(M', \rho')$, there is a simply-connected subset of this region, defined by ℓ and ρ', that the lift α' enters from left to right at the endpoint of θ'. Examples are drawn in Figure 4.11(c–d). Since curve α' cannot cross ρ' and crossing ℓ again would violate the left intersection property, α' ends in this region, and the first intersection of M and θ is normal.

We can further observe that the intersections of $M' \cap \theta'$ form a normal class, because the other end of θ also crosses ℓ and cannot return to M' without violating the left intersection property. This normal class corresponds to at most one special self-intersection $q \in \hat{\alpha}$: the portion of α intersecting M contracts to the unique vertical segment L, and the monotone chain of $\hat{\alpha}$ that contains M contains at most one point $q \in L$. Since the normal class $M' \cap \theta'$ does not contribute a normal self-intersection, the total number of normal and special self-intersections is bounded by the maximum number of normal classes, which is zero under the pin and pushpin definitions and $2n$ under the tack definition. This completes the proof. $\square$

We use an unrectified leftist path as our canonical path $\hat{\alpha}$ from now on.

4.2.6 Comparing canonical paths

Canonical sequences can have quadratic size, as the example of Figure 4.2 shows, so we still cannot compute the sequences for leftist canonical paths $\hat{\alpha}$ and $\hat{\beta}$. Fortunately, these paths can be compared by a more complicated version of the sweep algorithm for rectifying paths.

We continue to use notation introduced in the previous section. Path $\hat{\alpha}$ is a sequence of monotone chains delimited by turn points: let p_0 be the path starting point, p_m the path endpoint, and p_i the i^{th} turning point we encounter as we trace $\hat{\alpha}$. The i^{th} monotone chain α_i is defined as the portion of the path $\hat{\alpha}$ from p_{i-1} to p_i. We first state two lemmas that will allow us to compare canonical paths, then we describe a sweep algorithm to perform the comparison.

Lemma 4.8 *If the paths $\hat{\alpha}$ and $\hat{\beta}$ have the same canonical sequences, the corresponding monotone chains α_i and β_i are defined by the same two endpoints p_{i-1} and p_i.*

Proof: Each turning point p_i corresponds to rays from the same points and opposite direction ($l^+_{p_i} l^-_{p_i}$ or $l^-_{p_i} l^+_{p_i}$) in the canonical sequence. Since the path α and β have the same canonical sequences, the sequence of point label pairs with opposite bars are also the same. Therefore, the sequence of turning points are the same. By the definition of a monotone chain, each monotone chain α_i and β_i have the same endpoints. $\square$

Lemma 4.8 gives a necessary condition for $\hat{\alpha}$ and $\hat{\beta}$ to have the same canonical sequences: their monotone chains must have the same endpoints. If this is not satisfied, we can report that $\hat{\alpha}$ and $\hat{\beta}$ do not have the same canonical sequence, therefore α and β are not homotopic. Otherwise, further tests are needed.

We define *above sets* for points as follows: for each point p in the plane, let $A(\alpha, p) \subset \{1, 2, \ldots, m\}$ be the set of indices of monotone chains $\{\hat{\alpha}_i\}$ that lie above p. Similarly, let $A(\beta, p)$ be the set of indices of monotone chains $\{\hat{\beta}_i\}$ that lie above p.

Lemma 4.9 *If Lemma 4.8 is satisfied, then the paths $\hat{\alpha}$ and $\hat{\beta}$ have the same canonical sequence if and only if above sets $A(\alpha, p) = A(\beta, p)$ for each $p \in P$.*

Proof: To compare the canonical sequences, we can traverse the two paths $\hat{\alpha}$ and $\hat{\beta}$ with the same speed along the x-coordinate. Since the endpoints of each monotone chain $\hat{\alpha}_i$ are the same as $\hat{\beta}_i$, we pass the same sequence of points above or below in the plane as we traverse $\hat{\alpha}$ and $\hat{\beta}$. Suppose that monotone chains $\hat{\alpha}_i$ and $\hat{\beta}_i$ are being traversed (since their indices are both i) and we pass a point p. If the canonical sequences for $\hat{\alpha}$ and $\hat{\beta}$ are the same, the same ray from p (either l_p^+ or l_p^-) is in both sequences, so index i is either in both $A(\alpha, p)$ and $A(\beta, p)$ or in neither. Conversely, if $A(\alpha, p) = A(\beta, p)$, then p will have the same bar in the canonical sequences for $\hat{\alpha}$ and $\hat{\beta}$. □

Lemma 4.9, with one more idea, allows us to compare the canonical sequences using a sweep algorithm. The input of the algorithm is the set of monotone α-chains from leftist path $\hat{\alpha}$ and monotone β-chains from the leftist path $\hat{\beta}$. We sweep these chains from left to right with a vertical line. We maintain three invariants during the sweep, the most important being the *difference numbers* $D_x(k)$ defined below.

1. We maintain the set of monotone α-chains and the set of monotone β-chains intersected by the sweep line at position x.

2. Within each set, we maintain the aboveness order of the monotone chains and the rank of each chain in this order. It should be noted that the monotone chains could intersect, so this aboveness order depends on the current sweep line position x. (We keep the orders for $\hat{\alpha}$ and for $\hat{\beta}$ separate, otherwise we would need to compute all intersections between $\hat{\alpha}$ and $\hat{\beta}$.)

3. Finally, for sweep position x, let $A(\alpha, k)$ denote the *above set* of indices of monotone chains from $\hat{\alpha}$ that have ranks 1 through k. (That is, if we choose a point p on the sweep line just below the kth intersection with a monotone chain of $\hat{\alpha}$, then $A(\alpha, k) = A(\alpha, p)$.) We maintain the size of symmetric differences between $A(\alpha, k)$ and $A(\beta, k)$, and denote these *difference numbers* $D_x(k) = |\mathrm{diff}(A(\alpha, k), A(\beta, k))|$.

The invariants allow us to check the hypothesis of Lemma 4.9 to compare the canonical sequences. When the sweep encounters a point p, we locate p in the set of ordered monotone $\hat{\alpha}$-chains and the set of $\hat{\beta}$-chains. This gives us the sizes of the above sets, $|A(\alpha, p)|$ and $|A(\beta, p)|$. If these sizes are not equal, then the canonical sequences are not the same according to Lemma 4.9. Otherwise, we let k equal this size, and check if $D_x(k) = 0$. If it is not the case, $A(\alpha, p) \neq A(\beta, p)$, and Lemma 4.9 again tells us that the canonical sequences are not the same. If it is, we continue. If these conditions are satisfied by all $p \in P$, then the canonical sequences are the same by Lemma 4.9.

Having shown that the invariants allow us to compare canonical sequences, we give algorithms to maintain these invariants and establish their complexity. To maintain the aboveness order and the difference numbers, we keep these data structures:

- Two balanced binary search trees that use aboveness as the order: T_α stores the monotone α-chain indices, and T_β stores β-chain indices. Trees T_α and T_β allow us to search for the interval at which a point splits the monotone chains into above sets and below sets. By keeping counts in each subtree, we can also find the sizes of these sets.

- A balanced binary tree T_D that stores difference numbers $D_x(k)$ and uses k as the order. Difference numbers may be inserted or deleted.

We maintain these structures by handling events during a sweep, but let us first describe how to initialize these data structures for a given sweep position x.

Lemma 4.10 *The sweep data structures can be initialized for n chains at a sweep position x in $O(n \log n)$ time.*

Proof: We need to form T_α, T_β, and T_D, as defined above. For $\gamma \in \{\alpha, \beta\}$, sort the chains of γ by aboveness to form T_γ.

To form T_D, we must compute the difference numbers $D_x(k)$, for $k = 1 \ldots m$. First build two auxiliary tables: Let $I_\gamma(r)$ be the chain index at a given rank in T_γ and let $R_\gamma(i)$ be the rank of a given index; these are inverses: $I_\gamma(R_\gamma(i)) = i$ and $R_\gamma(I_\gamma(r)) = r$. Initialize $D_x(k) = 0$. Then, for $k = 1 \ldots m$, compute

$$D_x(k) = D_x(k-1) + \operatorname{sgn}\bigl(R_\beta(I_\alpha(k)) - k\bigr) + \operatorname{sgn}\bigl(R_\alpha(I_\beta(k)) - k\bigr),$$

where $\operatorname{sgn}(t)$ is 1 if t is positive, -1 if t is negative, and 0 if $t = 0$. This simply says that the difference number increases by one whenever the kth chain in the α list ranks higher in the β list and decreases when it ranks lower, and the same for the kth chain in the β list.

The most time-consuming step of this computation is sorting and ranking, which takes $O(n \log n)$ time for n chains. □

We maintain the data structure as we sweep the plane with a vertical line. At certain *events*, the sweep reaches the x-coordinate of a point at which we must update the data structures or compare the canonical sequences. In order to do this, the events clearly need to include monotone chain endpoints and points in the plane. Since the monotone chains from one path could intersect each other, the events also need to include all self-intersections of $\hat{\alpha}$ and $\hat{\beta}$. These intersections are computed during the sweep [18].

Before we describe how each event is handled, we make some simplifying assumptions about the monotone chains so that the algorithm is easier to describe—we will remove these assumptions later. First, we will treat each monotone chain as strictly monotone, even though chains from the canonical path may contain vertical line segments. Second, we will assume that no two event points have the same x-coordinate, even though this is not the case because of the vertical line segments. We consider five types of events.

(I) point in the plane: We check that the canonical sequences are still the same. First, locate the point among the monotone chains using T_α and T_β, then check that the difference number is zero, thereby verifying that the α above set is the same as the β above set.

(II) intersection event: Suppose the intersection is caused by monotone chains α_i and α_j so that α_i is below α_j after the intersection. To update T_α, we simply swap the monotone chain indices i and j. To update T_D, first find the aboveness rank k of α_i. Then, we know that the monotone chain α_i has deserted the aboveness set $A(\alpha, k)$; and the monotone chain α_j has entered $A(\alpha, k)$. Recall that the difference number $D_x(k) = |\text{diff}(A(\alpha, k), A(\beta, k))|$. So it is clear that $D_x(k)$—and only $D_x(k)$ in T_D—needs to be changed. To change $D_x(k)$ appropriately, we find out whether the monotone chain β_i is in $A(\beta, k)$. If it is, then increment $D_x(k)$. Otherwise, do nothing. Symmetrically, if the monotone chain β_j is in $A(\beta, k)$, decrement $D_x(k)$. Otherwise, do nothing.

(III) path endpoint: At a path endpoint, which must be shared by the α chain and β chain, we use Lemma 4.10 to reinitialize the data structures.

(IV) chain starting event: At a turn point, we may have m monotone chains that begin in α and in β. We compute the difference numbers restricted only to the m new chains, which we denote $D'_x(r)$, for $r = 1 \ldots m$. This can be done using Lemma 4.10 restricted to these m new chains.

To update the algorithm's data structures, we next locate the event point in the trees T_α and T_β and then insert the new chains in these trees in the aboveness order. Because the new chains in α and β start at the same turning point, they are adjacent in their respective trees, and we can insert them efficiently.

We must now update the difference numbers in T_D. Note that because the event is a turning point, if the insertion does not take place at the same rank k in both T_α and T_β, or if different numbers of α and β chains are inserted, then the paths are not homotopic by Lemma 4.9. Thus, we need only insert m new differences; the old difference numbers do not otherwise change. Since the r^{th} interval in the new monotone chains becomes the $(r + k)^{th}$ interval after the insertion, we insert the new difference numbers $D_x(r + k) = D'_x(r) + D_x(k)$ for $r = 1, 2, \ldots, m$ into T_D.

(V) chain ending event: The operations for updating data structures at a turning point where m monotone chains end are exactly the reverse of the operations at a chain starting event. To update T_α and T_β, we delete the indices of all monotone chains ending at the event point. Assuming that these had ranks $k + 1$ through $k + m$, we delete the difference numbers $\{D_x(k + 1), D_x(k + 2), \ldots, D_x(k + m)\}$ from T_D.

The number of intersection events (type II) depends on the definition of path that we use, but we have shown a linear upper bound in the previous section. Thus, our sweep to compare canonical sequences is efficient for all definitions of homotopy that we described.

Lemma 4.11 *Linear space and $O(n \log n)$ time are sufficient to compare the canonical sequences of canonical paths $\hat{\alpha}$ and $\hat{\beta}$, assuming that the number of self-intersections in each canonical path is $O(n)$.*

Proof: The space is linear since all the data structures we used are linear (T_α, T_β and T_D). To bound the time, note that each operation to maintain T_α and

T_β (insertion or deletion of the chain indices) can be charged to a monotone chain endpoint, and each operation for maintaining T_D (computing, insertion or deletion of the difference numbers) can be charged either to a monotone chain endpoint or an intersection point. Since the time for each of these operations is at most $O(\log n)$, and we assume that there are $O(n)$ monotone chain endpoints and intersection points, the total time is $O(n \log n)$. $\square$

By careful implementation, we can get rid of the assumptions that simplified the algorithm description. First, we assumed that the chains were strictly monotone, but in fact they contain vertical line segments. Therefore, in order for intersection points to be handled correctly, the sweep line should stop at a vertical line segment, compute the intersections on it and order these intersection events by the order they are encountered by the path. Second, we assumed that no two events have the same x-coordinate, although some points in the plane, such as the turning points, lie on vertical line segments. To handle these points correctly, we need to perturb them away from the vertical line segments in the right direction: from the canonical path computation, we know whether the line segment is some path contracted from the right side of the point or from the left. If it is from the left, this event should be handled immediately after the vertical line segment; if is from the right, it should be handled immediately before the vertical line segment. We summarize.

Theorem 4.12 *Given two simple paths α, β, each consisting of n segments, and a set P of n points, we can decide if α and β are homotopic in the plane minus P in $O(n \log n)$ time and linear space. This holds for the tack, pin, and pushpin definitions of homotopic paths.*

4.3 Homotopy test for non-simple paths

When paths are allowed to intersect themselves, as well as each other, then we cannot use aboveness to rectify the paths as in the previous section. In this section we show that we can use a different construction of a universal cover and achieve a running time near $O(n^{3/2})$.

Since the paths may self-intersect, we simply concatenate them as suggested in the introduction and consider whether a closed loop α is contractible in the plane minus P. The idea is to lift the path into a universal cover constructed in an appropriate way, and test if the endpoints of the lifted path coincide. We will use a simple path d going through all points in P in order to construct such a universal cover, but, in order to minimize the number of operations during the lifting of α, we have to minimize the number of intersections between d and α. The way to construct d is using a spanning tree with low crossing number, that is, a spanning tree T of the point set P such that any line intersects T at most $O(\sqrt{n})$ times.

Lemma 4.13 *A simple spanning tree of P with crossing number $O(\sqrt{n})$ can be constructed in $O(n^{1+\epsilon})$ time, for any $\epsilon > 0$.*

Proof: We combine ideas and results from [59, 78, 98, 99]. Start by computing a spanning tree T_1 of P with crossing number $O(\sqrt{n})$. Matoušek shows in [99] how a simplicial partition $\{P_1, \ldots, P_{n/2}\}$ with $|P_i| \leq 4$ and crossing number $O(\sqrt{n})$ can be constructed in time $O(n^{1+\epsilon})$. This partition plays the role of the partial matching used in [98]: for each class P_i with $|P_i| \geq 2$, we put an edge between a pair of points lying in P_i, and remove either of its endpoints from the set of points P. Iterating this step $O(\log n)$ times, we get the spanning tree T_1 within $O(n^{1+\epsilon})$ operations.

Tree T_1 can have self-intersections, but we can use it to construct a Steiner spanning tree T_2 that is simple [59]. We start with an empty tree T_2. Then, we traverse T_1 in preorder. When we visit a vertex $p \in P$, we shoot a ray along the edge connecting p toward its predecessor q in the preorder of T_1. Then we add to T_2 the segment from p to the first intersection of the ray with the current T_2. Note that this first intersection may be q. The dynamic data structure for ray shooting described in [78] supports updates and queries in $O(\log^2 n)$ time, so T_2 can be constructed in $O(n \log^2 n)$ time. Observe that T_2 has $O(n)$ edges, and that, as subset of T_1, it also has crossing number $O(\sqrt{n})$.

As proved in [59], from T_2 we can construct in $O(n \log n)$ time a spanning tree T_3 with no Steiner points, and whose crossing number is twice that of T_2. $\square$

Theorem 4.14 *We can decide if a closed loop α is contractible in the plane minus P in $O(n^{3/2} \log n)$ time.*

Proof: Take a bounding box B enclosing α and P, and let p_- and p_+ be two points on the left and on the right of B, respectively. We apply the previous lemma to the set $P \cup \{p_-, p_+\}$ to obtain a simple spanning tree, and then we use it to make a path starting at p_- and finishing at p_+. If we perturb some edges of this path slightly, we can convert it into a simple path d from p_- to p_+, passing through all points, and having crossing number $O(\sqrt{n})$.

This path d splits B into two pieces, B_0 and B_1, which we can use as building blocks for the portion of a universal cover that contains the lift of the loop α. We actually build the dual graph T (a tree) for the universal cover, and keep just one copy of each block. Thus, we can represent the portion of the universal cover containing the lift of α in space linear in n plus the number of nodes of the dual tree T.

The construction of T is as follows. Preprocess blocks B_0 and B_1 for ray shooting [86]. Start with some vertex of α in B_i, where $i \in \{0, 1\}$, and create a tree node ν to represent this copy of B_i.

Now, use ray shooting along the segments of α to trace α in B_i until it leaves by crossing an edge e of the path d. If edge e has never been crossed before, then make a new node η in the dual tree T, put an edge between current node ν and η, annotated with the edge e. If edge e has been crossed before, then ν has an edge annotated with e that connects to a node η. In either case, update the current node $\nu = \eta$, and set $i = i + 1 \bmod 2$ to switch to the other block.

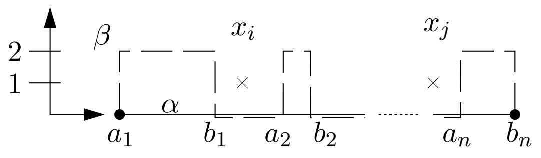


Figure 4.13: Lower bound for simple paths.

When we complete the tracing, we know that α is contractible if and only if we ended up in the same node of T in which we started.

Because each edge of T comes from one intersection of d and α, and given that the crossing number of d is $O(\sqrt{n})$, tree T has size $O(n^{3/2})$. An additional logarithmic factor in the running time is sufficient to pay for the ray shooting. □

4.4 Lower bounds

In this section we establish lower bounds for the path homotopy problem among points in the plane. The bounds are different depending on whether the paths are simple or not.

4.4.1 Simple paths

When both paths α and β are simple, to decide if they are homotopic takes a minimum of $\Omega(n \log n)$ operations in the decision tree model. In order to show this, consider the following problem: given a set of n unsorted numbers $x_1, \ldots,$ x_n, and a set of n disjoint intervals $[a_1, b_1], \ldots, [a_n, b_n]$, in increasing order, is there any interval containing any point? Using Ben-Or's theorem [17], it is straightforward to see that in the algebraic decision tree model, this problem has a lower bound of $\Omega(n \log n)$.

This problem can be reduced to testing whether two simple paths are homotopic. Consider α to be the segment with endpoints $(a_1, 0)$ and $(b_n, 0)$, and β to be the path following α except when it overlaps with some interval $[a_i, b_i]$, in which case it will follow the horizontal segment at height 2, as shown in Figure 4.13. If we consider the set of points $P = \{(x_i, 1) \mid 1 \leq i \leq n\}$, then no point belongs to any interval if and only if the paths α and β are homotopic.

Theorem 4.15 *Given two simple paths α, β, each consisting of n segments, and a set P of n points, any algorithm in the algebraic decision tree model needs $\Omega(n \log n)$ time to decide if α and β are homotopic in the plane minus P.*

4.4.2 Non-simple paths

When the polygonal lines α and β are allowed to self-intersect, then the problem becomes computationally harder: we can reduce Hopcroft's problem to the problem of testing if two paths are homotopic.

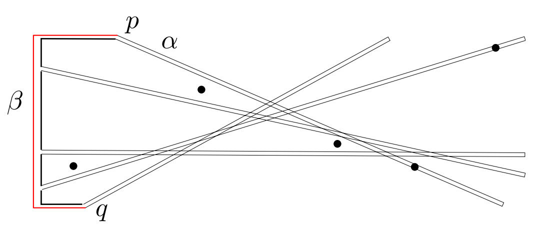


Figure 4.14: Reduction from Hopcroft's problem.

Given n points and n lines, *Hopcroft's problem* asks if any point lies on any line. The best algorithms known to solve Hopcroft's problem take just slightly more than $O(n^{4/3})$ time [44]. Erickson showed that *partition algorithms*, a certain class of algorithms that includes the natural and known algorithms, take at least $\Omega(n^{4/3})$ time [68].

We give a reduction that works for any algorithm whose primitive tests check the signs of bounded-degree polynomials of the input point coordinates, which includes partition algorithms. We need this restriction so that we can use infinitesimal quantities in the construction. As is common in perturbation methods, we can expand the primitives as polynomials in one infinitesimal variable and determine the sign of the term of smallest degree [60].

Given an instance of Hopcroft's problem, determine a bounding box whose top (bottom) side is higher (lower) than all the vertices of the arrangement of lines. Let p and q be points on the bounding box that lie in these faces, as illustrated in Figure 4.14. This can be done in linear time after sorting the lines by slope. Let β be the left path along the bounding box from p to q. Let α be the path that follows β except for leaving the bounding box at every line, crossing the box infinitesimally above the line, and returning infinitesimally below the line to β. Paths α and β are homotopic if and only if no point lies on any line.

Theorem 4.16 *Given two paths α, β, each consisting of n segments, and a set P of n points, any partition algorithm needs $\Omega(n^{4/3})$ time to decide if α and β are homotopic in the plane minus P.*

4.5 Concluding remarks

We have given an efficient algorithm to test homotopy for simple paths in the plane by using an aboveness ordering to rectify paths. We believe that it may be possible to eliminate the range queries by adding segments to the path in aboveness order instead of traversal order, but each way we have tried so far slips back to a quadratic worst case running time. For non-simple paths, we use standard machinery to give a subquadratic algorithm. Lower bounds show that the running time of our first algorithm is tight. The algorithm for the non-simple case has been recently improved by Bespamyatnikh [21] with an algorithm that matches our lower bounds up to a polylogarithmic factor.

Chapter 5

Schematic maps

This chapter presents efficient algorithms for schematic map construction. The input we assume is a planar embedding of a graph consisting of polygonal paths between specified points called endpoints. It represents, for instance, a road or railroad network. The output should consist of another planar embedding where all endpoints have the same positions, and every path is displayed as a two-link or three-link path where links are restricted to certain orientations (Figure 5.1). Furthermore, the output map should be equivalent to the input map in the sense that a continuous deformation exists such that no path passes over an endpoint during the transformation. This topological equivalence can also be expressed in the following way: our algorithm keeps the face structure in both embeddings. A consequence of this equivalence is that cyclic order of paths around endpoints is maintained. If a schematized map of the specified type does not exist, our algorithm reports this failure.

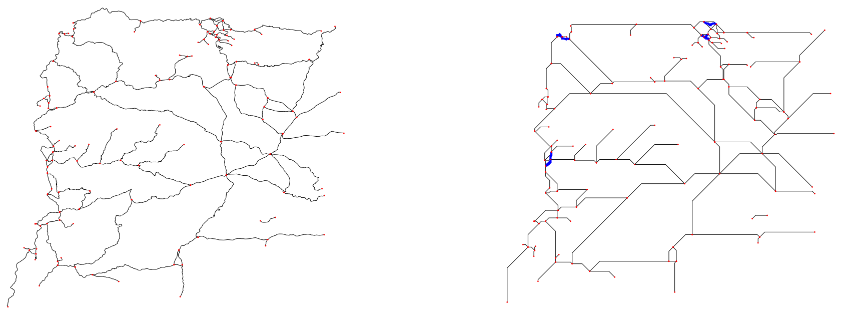


Figure 5.1: Northwest of Iberic Peninsula. Left: original map. Right: the schematized version made by the implementation.

In our approach, we assume that the types of schematic paths that are allowed in the schematic map are fixed a priori. We cannot handle all types, but subclasses that have some sort of well-ordering property. Schematic types with this property include, for example, axis-parallel paths with at most two links each, or paths with links in the four main orientations (axis-parallel, and angles of $45°$ and $135°$). A vertical minimum separation distance between schematic paths can be specified, and in certain cases we can allow or disallow that paths leaving the same endpoint partially share the first link. Details are given in Section 5.3. When the original map consists of n segments, our algorithm runs in $O(n \log n)$ time, and experimental results show the quality of the output.

The rest of this chapter is structured as follows. Section 5.1 describes the basic definitions and concepts. We extend the concept of equivalent paths (Definition 4.1) to equivalent maps. Furthermore, we define an order between the paths of a map and we discuss its basic implications.

Section 5.2 explains how to compute the order introduced in Section 5.1 among the paths of the input map. Globally, our algorithm works as follows. Similar to Section 4.2 (see also [64]), we use an aboveness order among monotone pieces of the paths and we bring our problem into an orthogonal setting. While keeping homotopy type and simplicity, we simplify each path to an x-monotone one. Then, we show that the order among paths in this map and in the original one are the same, and we use the map with x-monotone paths to compute this order.

In a map, several paths may share a common endpoint, and this makes a difference with respect to the previous chapter. Furthermore, if we directly use the results of the previous chapter to simplify each path independently, then we get quadratic running time if we have a linear number of endpoints (which play the role of obstacles). Therefore, although most of the ideas that we present in Sections 5.1 and 5.2, have already appeared in the previous chapter, we also make a careful presentation here.

Section 5.3 describes the method to place the schematized paths. We place each path following the computed order from top to bottom. Each path is placed at the topmost position that is still possible. This will leave the maximum freedom for later paths that need to be placed. We can specify the type of schematized paths allowed and a vertical separation distance between paths, if desirable. If in the transformed map of Section 5.2 there is no order (a cycle is detected) among its paths, or the placement of some schematized path fails, then a schematized map of the desired type does not exist.

Section 5.4 shows some experimental results in order to evaluate the visual quality of the output provided by our algorithm; see Figure 5.1. Finally, Section 5.5 gives a summary and directions for further research.

5.1 Equivalent maps: definition and basic properties

5.1.1 Equivalent paths and equivalent maps

Like in the previous chapter, a path is a continuous mapping $\alpha : [0,1] \to \mathbb{R}^2$, and the path is simple (no self-intersections) if the mapping is injective. We use $|\alpha|$ to denote the complexity of path α, defined as the number of edges it has.

Definition 5.1 *A* map *M is a set of simple polygonal paths $\{\alpha_1, \ldots, \alpha_m\}$ such that two paths do not intersect except at shared endpoints. A* monotone map *is a map where all paths are x-monotone.*

We use P_M to denote the set of endpoints $\{\alpha(0), \alpha(1) \mid \alpha \in M\}$, and $n = \sum_{i=1}^{m} |\alpha_i|$ for the complexity of the whole map. To simplify presentation we assume that all vertices in the map have different x-coordinates; our algorithm can be adapted in a straightforward way for the general case.

Our goal is to construct schematized versions of a given map that are equivalent to it. Intuitively, two maps are equivalent if we can transform all paths from one map to the other one in a continuous way, fixing the endpoints and without crossing any 'important point'. For example, if a road passes to the north of an important city, we do not want the schematized version to pass to the south of that city. Furthermore, we want the cyclic order of roads at crossings to stay the same. We let the important points for one path be the endpoints of the other paths; our algorithms can easily be adapted to take additional important points into account.

To formalize the approach and its properties we use the concept of equivalent paths (see Definition 4.1 and the discussion that follows it). For cartographic purposes, it seems better to consider the pin definition of equivalent maps.

Definition 5.2 *Two maps $M = \{\alpha_1, \ldots, \alpha_m\}$ and $\tilde{M} = \{\beta_1, \ldots, \beta_m\}$ are* equivalent maps *if and only if for some renumbering of the paths in $\tilde{M}$, paths α_i and β_i are equivalent in $(\mathbb{R}^2 \setminus P_M) \cup \{\alpha_i(0), \alpha_i(1)\}$ for all paths $\alpha_i \in M$.*

The problem of schematizing a map can now be restated as follows: given a map, compute an equivalent map whose paths are of a certain type (such as axis-aligned, x-monotone, 3-links, etc.) and have certain properties (such as a minimum vertical distance between two schematized paths).

5.1.2 Order among paths

For a point p, we use l_p^+ and l_p^- to denote the vertical halfline with point p as lowest and highest point respectively. We next define aboveness of paths, which is an invariant among equivalent paths and equivalent maps.

Definition 5.3 *Let α be a path. A point $p \in \mathbb{R}^2$ with $p \notin \alpha$ is* above (below) *α if for every equivalent path β in $\mathbb{R}^2 \setminus \{p\}$ the intersection of l_p^- (respectively*

l_p^+) and β is nonempty. A path α_i is above *α_j if $\alpha_i(0)$ or $\alpha_i(1)$ is above α_j, or if $\alpha_j(0)$ or $\alpha_j(1)$ is below α_i.*

We would like to remark that, in this definition, to decide whether a point is above or below a path, we do not take into account any other point, that is, our universe is reduced exclusively to the point and the path. Observe that in some cases, it is possible that a point is both above and below a curve. In other cases, no order is present. See Figure 5.2 for an example.

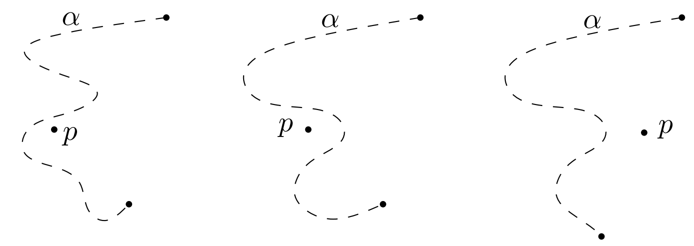


Figure 5.2: Example of relations between points and paths. Left: p is both above and below α. Center: p and α have no relation. Right: p is below α.

Lemma 5.4 *The above-below relation among paths is invariant between equivalent maps.*

Proof: The equivalence of paths is an equivalence relation [43, 82]. For this reason, for any point $p \in \mathbb{R}^2$, equivalent paths in $\mathbb{R}^2 \setminus \{p\}$ have the same relation with respect to the point p.

Let M and $\tilde{M}$ be equivalent maps. For any path $\alpha_i \in M$, let $\beta_i \in \tilde{M}$ be its corresponding path. Because α_i and β_i are equivalent in $(\mathbb{R}^2 \setminus P_M) \cup \{\alpha_i(0), \alpha_i(1)\}$, it holds that for any endpoint $p \in P_M \setminus \{\alpha_i(0), \alpha_i(1)\}$ they are also equivalent in $\mathbb{R}^2 \setminus \{p\}$. So, any endpoint different from $\alpha_i(0), \alpha_i(1)$ has the same above-below relation with α_i and β_i, and this implies the result. $\square$

From an algorithmic point of view, it is not clear how Definition 5.3 can be used to decide if a point $p \in P$ has some above-below relation with a path α. The canonical sequences introduced in Section 4.1.2 can handle this issue.

Consider the canonical sequence of α for the particular case when P consists of only one point p. Any path β equivalent to α in $\mathbb{R}^2 \setminus \{p\}$ has to cross the rays l_p^+ or l_p^- that appear in the canonical sequence of α with respect to $\{p\}$. Thus, a point p is below (above) a path α if and only if l_p^+ (respectively l_p^-) appears in the canonical sequence of α with respect to $\{p\}$. See Figure 5.3, right, for an example.

5.1.3 Above-below relations in monotone maps

In Section 4.2.1 the following aboveness order between x-monotone paths A and B has been introduced: $A \succ B$ if and only if there are points $(x, y_A) \in A$ and

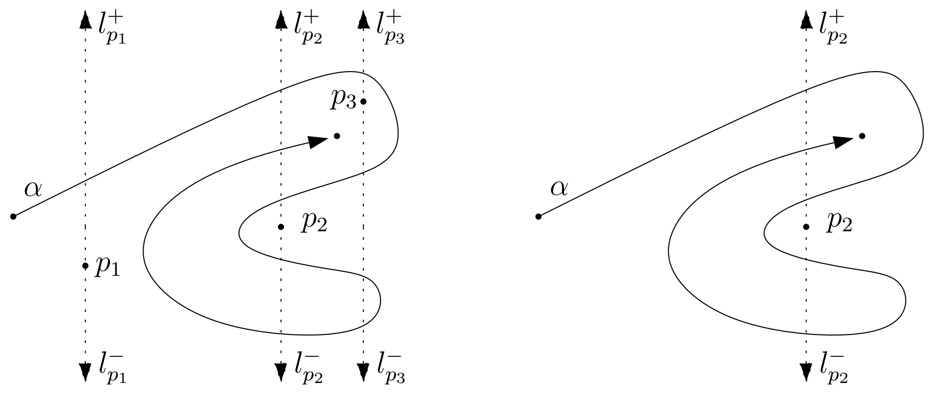


Figure 5.3: Left: example of a canonical sequence. The original sequence is $l^+_{p_1}l^+_{p_2}l^+_{p_3}l^-_{p_3}l^+_{p_2}l^-_{p_2}l^-_{p_3}l^-_{p_3}l^-_{p_2}l^+_{p_2}$ and thus its canonical one is $l^+_{p_1}l^+_{p_2}l^+_{p_3}l^-_{p_3}$. Right: example comparing p_2 with α. The sequence to consider is $l^+_{p_2}l^+_{p_2}l^-_{p_2}l^-_{p_2}l^+_{p_2} \equiv l^+_{p_2}$, and we can conclude that p_2 is below path α.

$(x, y_B) \in B$ with $y_A > y_B$. It is important to note that this relation is the same as the one given in Definition 5.3 for the case of disjoint, x-monotone paths. We restate Lemma 4.4 and the discussion that follows in this context.

Lemma 5.5 *For a simple, monotone map M, the above-below relation among paths is acyclic. Furthermore, if M has complexity n, a total order extending this relation can be computed in $O(n \log n)$ time.*

Corollary 5.6 *For a given map M, if there is no partial order among its paths, then no monotone map can be equivalent to M.*

Proof: This follows from Lemma 5.4 and Lemma 5.5. □

Because we are only interested in schematic maps with x-monotone paths, this corollary implies that the algorithm to be developed may report the impossibility of the schematic map. For monotone maps, the above-below relation provides a complete characterization up to equivalence, which is shown in the next lemma.

Lemma 5.7 *Let $M = \{\alpha_1, \ldots, \alpha_m\}$ and $\tilde{M} = \{\beta_1, \ldots, \beta_m\}$ be two monotone maps such that the paths α_i and β_i have the same endpoints. Then, the maps M and $\tilde{M}$ are equivalent if and only if they define the same above-below relation.*

Proof: One implication is provided by Lemma 5.4. For the other implication, we show that if M and $\tilde{M}$ are not equivalent, then they define a different above-below relation. Observe that if α_i and β_i are not equivalent in $(\mathbb{R}^2 \setminus P_M) \cup \{\alpha_i(0), \alpha_i(1)\}$, it is due to the existence of some point $p \in P_M \setminus \{\alpha_i(0), \alpha_i(1)\}$ in the regions between α_i and β_i (these regions are well-defined because both α_i and β_i are x-monotone; see Figure 5.4 left). It follows that p has a different above-below relation with α_i and β_i, and for any path α_j with p as an endpoint, the relation between paths α_j and α_i will be different from the relation between β_j and β_i. □

Observe that this result is not true if we remove the monotonicity conditions, as shown in Figure 5.4 right.

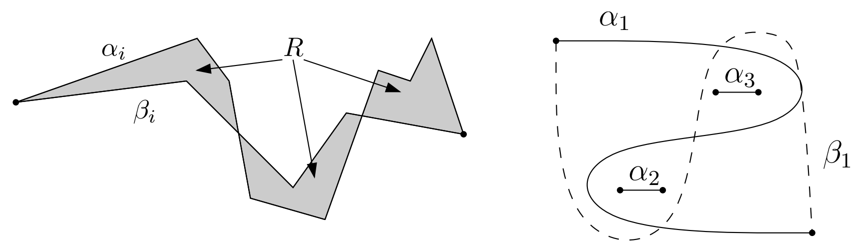


Figure 5.4: Left: two x monotone paths α_i, β_i that share the same endpoints have a well-defined region R between them (gray in figure). If there is a point p inside R, then α_i and β_i cannot be equivalent in $\mathbb{R}^2 \setminus \{p\}$. Right: maps $\{\alpha_1, \alpha_2, \alpha_3\}$ and $\{\beta_1, \alpha_2, \alpha_3\}$ define the same above-below relations but they are not equivalent.

5.2 Computing order in a map

The straightforward approach of comparing each pair of paths to decide their relation gives a worst-case quadratic time algorithm. In this section we show how to compute a total order extending the partial order defined in the previous section in $O(n \log n)$ time. Since sorting real numbers can be reduced to computing the order in a map, our approach is asymptotically optimal. The basic ideas explained here are taken from the previous chapter, but we have to use them carefully because many paths may share endpoints, which makes some of the argumentations more involved. Firstly, we convert the path into what is called a rectified map. Secondly, we transform the rectified map into one with x-monotone pieces, and, finally, we compute the order in this map.

5.2.1 Rectified maps

Definition 5.8 *A set of paths $M' = \{\alpha'_1, \ldots, \alpha'_m\}$ is a* rectification *of a map $M = \{\alpha_1, \ldots, \alpha_m\}$ if:*

- *M' is a map (its paths only intersect in common endpoints);*
- *the complexity of map M' is linear in the complexity of map M;*
- *paths α'_i are made of axis-aligned segments;*

- *paths α_i and α_j have the same above-below relation as α'_i and α'_j.*

A map $M = \{\alpha_1, \dots, \alpha_m\}$ can be rectified in the following way. Decompose each path $\alpha_i \in M$ into monotone pieces $\alpha_i^1, \dots, \alpha_i^{k_i}$. By promoting every locally leftmost or rightmost vertex of a path to an endpoint, we make a monotone map M_{mono} with the set of pieces $\{\alpha_1^1, \dots, \alpha_1^{k_1}, \dots, \alpha_m^1, \dots, \alpha_m^{k_m}\}$ as the new set of paths. Because M_{mono} has complexity $O(n)$, we can compute—among the pieces α_i^j—a total order extending the partial order in $O(n \log n)$ time (Lemma 5.5).

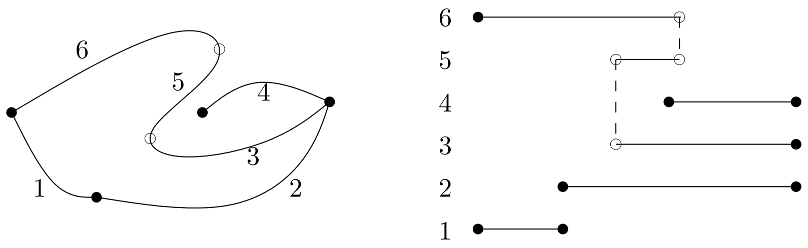


Figure 5.5: A map with its paths partitioned into monotone pieces, and their ranks. Right, the corresponding rectified map.

Using the rank of each monotone piece in the total order, we can construct the rectified version of a path as follows; see Figure 5.5. Let α_i^j be a monotone piece with left endpoint (p_x, p_y), right endpoint (q_x, q_y), and rank r. Then, we make the horizontal segment $h_i^j = [p_x, q_x] \times r$ in the rectified map, and we form the path α'_i that joins $h_i^1, \dots, h_i^{k_i}$ by connecting the endpoints of every two consecutive horizontal segments by a vertical segment. We denote by M' the collection of all such paths, that is, $M' := \{\alpha'_1, \dots, \alpha'_m\}$. Observe that an endpoint sharing several paths will be mapped in several points vertically above each other in M'.

Lemma 5.9 *Given a map M of complexity n, we can construct a rectification M' in $O(n \log n)$ time.*

Proof: The computation of M' as described takes $O(n \log n)$ time if we use Lemma 5.5. It is also clear from the construction that M' satisfies the first three conditions to be a rectification of M. It remains to show that it also fulfills the fourth condition.

Consider the canonical sequence s of the path α_i with respect to the endpoint $\alpha_j(0)$ and the canonical sequence s' of the path α'_i with respect to $\alpha'_j(0)$. From the construction of M', it follows that, replacing in the sequence s each occurrence of $l^+_{\alpha_j(0)}$ (respectively $l^-_{\alpha_j(0)}$) by $l^+_{\alpha'_j(0)}$ (respectively $l^-_{\alpha'_j(0)}$), we get the sequence s'. Since a canonical sequence is obtained by removing adjacent elements that are identical, we conclude that the above-below relation of $\alpha_j(0)$ to α_i is identical to the above-below relation of $\alpha'_j(0)$ to α'_i. The same argument applies to the relation of $\alpha_j(1)$ to α_i and the relation of $\alpha_i(0), \alpha_i(1)$ to α_j, and thus the above-below relation of α_i and α_j is the same as that of α'_i and α'_j. $\square$

5.2.2 Computing order using a rectified map

Let $M' = \{\alpha'_1, \ldots, \alpha'_m\}$ be a map such that all segments are axis-aligned. In Section 4.2.4 we have introduced the algorithm Rcp that transforms the path α'_i into another path $\textsc{Rcp}(\alpha'_i)$. Let us summarize the properties of $\textsc{Rcp}(\alpha'_i)$ that we will use:

1. $\textsc{Rcp}(\alpha'_i)$ is equivalent to α'_i in $(\mathbb{R}^2 \setminus P_{M'}) \cup \{\alpha'_i(0), \alpha'_i(1)\}$;
2. $|\textsc{Rcp}(\alpha'_i)| \leq |\alpha'_i|$;
3. if α'_i and α'_j do not intersect, and $\textsc{Rcp}(\alpha'_i)$, $\textsc{Rcp}(\alpha'_j)$ are monotone, then $\textsc{Rcp}(\alpha'_i)$ and $\textsc{Rcp}(\alpha'_j)$ do not intersect either;
4. $\textsc{Rcp}(\alpha'_i)$ has the minimum possible number of x-monotone pieces that any path equivalent to α'_i in $(\mathbb{R}^2 \setminus P_{M'}) \cup \{\alpha'_i(0), \alpha'_i(1)\}$ can have.

To use Rcp, we first have to preprocess the endpoints $P_{M'}$ of the map M' for the three-sided range queries that were discussed in Section 4.2.3. This can be done in $O(|P_{M'}| \log |P_{M'}|)$ time, and then, it takes $O(|\alpha'_i| \log |P_{M'}|)$ time to compute $\textsc{Rcp}(\alpha'_i)$ because of Theorem 4.6. Therefore, computing $\textsc{Rcp}(\alpha'_i)$ for all paths $\alpha'_i \in M'$ takes

$$O(|P_{M'}| \log |P_{M'}|) + \sum_{\alpha'_i \in M'} O(|\alpha'_i| \log |P_{M'}|) =$$

$$= O(|M'| \log |P_{M'}|) = O(|M'| \log |M'|).$$

This is the basic ingredient for the main result of this section.

Theorem 5.10 *For a map M of total complexity n, we can decide in $O(n \log n)$ time whether an equivalent, simple, monotone map exists.*

Proof: Given the map $M = \{\alpha_1, \ldots, \alpha_m\}$, we start by making a rectification of it, $M' = \{\alpha'_1, \ldots, \alpha'_m\}$, and then we use Rcp to compute $N := \{\textsc{Rcp}(\alpha'_1), \ldots, \textsc{Rcp}(\alpha'_m)\}$. Observe that by Definition 5.8, Lemma 5.9, and the first two properties of Rcp, it follows that N has complexity $O(n)$, it can be constructed in $O(n \log n)$ time, and it has the same above-below relations among its paths as M does. We claim that N is a monotone map if and only if M admits an equivalent, monotone map.

Recall that a point $p \in P_M$ that is an endpoint of several paths in M, is mapped into several endpoints in M'. Let p' be this set of endpoints. By construction, no segment of M' can pass between two points of p', and when considering the canonical sequences of a path α'_i with respect to $P_{M'} \setminus \{\alpha'_i(0), \alpha'_i(1)\}$, we can just consider all points in p' as one point, which we also denote by p'. With this notation, if we replace in the canonical sequence of α_i with respect to $P_M \setminus \{\alpha_i(0), \alpha_i(1)\}$ each occurrence of l_p^+ by $l_{p'}^+$ and l_p^- by $l_{p'}^-$, where p is any point in $P_M \setminus \{\alpha_i(0), \alpha_i(1)\}$, then we get the canonical sequence of α'_i with respect to $P_{M'} \setminus \{\alpha'_i(0), \alpha'_i(1)\}$.

If M admits an equivalent, monotone map, then for each path $\alpha_i \in M$, the canonical sequence of α_i with respect to $P_M \setminus \{\alpha_i(0), \alpha_i(1)\}$ does not contain two rays emanating from the same point. But this is equivalent to saying that the canonical sequence of α'_i with respect to $P_{M'} \setminus \{\alpha'_i(0), \alpha'_i(1)\}$ does not contain two rays emanating from the same point, which, by property 4 of RCP, implies that RCP(α'_i) is x-monotone. Together with property 3, this implies that N is a monotone map.

Conversely, if N is a monotone map, then for each path $\alpha_i \in M$ there is a path β_i that is x-monotone and equivalent to α_i in $(\mathbb{R}^2 \setminus P_M) \cup \{\alpha_i(0), \alpha_i(1)\}$. Consider the collection of paths $\{\beta_1, \ldots, \beta_m\}$. They do not need to be a map, but we will show how to convert it into a monotone map that is equivalent to M. Let β_i and β_j be paths that intersect, which implies that they do so at least twice (tangencies and common endpoints do not count as intersections): if β_i and β_j would intersect only once, then β_i would be both above and below β_j (see Figure 5.6 left), but then also RCP(α'_i) would be above and below RCP(α'_j), which is not possible because N is a monotone map by hypothesis. On the other hand, no point $p \in P_M$ can lie in the region that β_i, β_j define between their first and second intersection (see Figure 5.6 right), otherwise β_i would be both above and below β_j. Therefore, we can deform β_i and β_j into other paths (we also use β_i, β_j for the new paths) and we reduce the total number of intersections by two (see Figure 5.6 right). The new path β_i is also equivalent to α_i in $(\mathbb{R}^2 \setminus P_M) \cup \{\alpha_i(0), \alpha_i(1)\}$, and the similar statement holds for β_j. Each time we apply this process we reduce the number of intersections in $\{\beta_1, \ldots \beta_m\}$, and thus if we repeat this process for each pair of paths that intersect, we will eventually end up with a set of x-monotone paths that do not have intersections (except at shared endpoints). This is a monotone map that is equivalent to the original one. $\square$

Corollary 5.11 *If for a map M there is an equivalent, monotone map, then we can compute a total order extending the partial order among its paths in $O(n \log n)$ time.*

Proof: The order among paths in map $M = \{\alpha_1, \ldots, \alpha_m\}$ is the same as the order in map $N = \{\text{RCP}(\alpha'_1), \ldots, \text{RCP}(\alpha'_m)\}$. The hypothesis implies that N is monotone, and by Lemma 5.5 we can therefore compute such a total order. $\square$

5.3 Placing paths in the schematic map

In the previous sections we determined a partial top-to-bottom order on the paths in the input map. This section concentrates on the actual placement of the paths: we show that placing the paths one by one, in the computed order, where each path is placed topmost (that is, as high as possible) will result in a schematic map if one exists. We use the notation $X_1X_2X_3$-path (X_1X_2-path), with $X_i \in$ {H, V, D} to denote a 3-link (respectively 2-link) path whose i-th

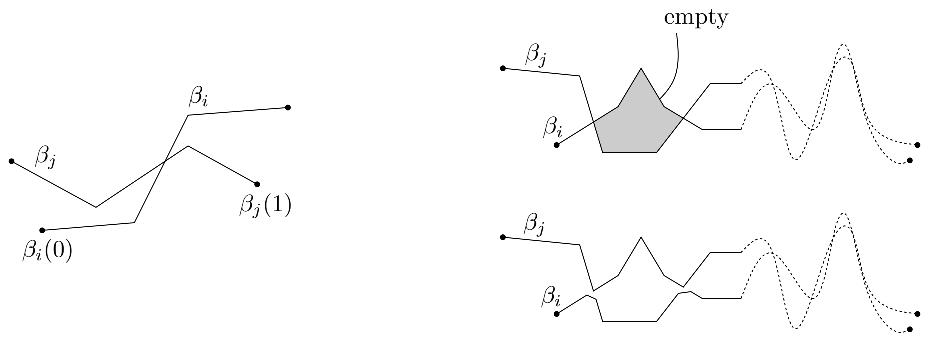


Figure 5.6: Left: if the paths β_i and β_j only intersect once, then they have a cyclic above-below relation. In this example, because the endpoint $\beta_i(0)$ is below β_j, the path β_i is below β_j, but also $\beta_j(1)$ is below α_i, and so β_j is below β_i. Right: if β_i and β_j intersect more than once, the region between the first two intersections is well-defined because they are x-monotone (grey in the picture). This region cannot contain any point $p \in P_M$, otherwise β_i is above p which is above β_j, and we get a cyclic order. Then we can remove the intersection continuously in $(\mathbb{R}^2 \setminus P_M) \cup \{\alpha_i(0), \alpha_i(1)\}$.

link is of type X_i. Here, type H means horizontal, type V means vertical, and type D means diagonal.

To explain the main features of the placement we will describe the algorithm for one concrete case: 3-link {HDH, VDV}-paths that are L_2-shortest. This type of paths is shown in Figure 5.7 left and center, and a map using this type of paths is in Figure 5.8 (a). Later we will generalize to other types of paths. The idea is to incrementally place the paths from top to bottom, respecting the order, and maintaining the lower envelope of the previously placed paths in a binary search tree $\mathcal{T}$. The tree stores in its leaves, ordered from left to right, the vertices and segments of the lower envelope by increasing x-coordinate. When adding a path, we search with the left and right endpoints in $\mathcal{T}$, ending in two leaves μ and μ'. We collect the part of the lower envelope in between. The new path must be below the collected pieces of the lower envelope. If one of the endpoints of the new path is above the lower envelope stored at μ or μ', the algorithm fails and no schematization exists. Otherwise, we can determine the topmost placement of the new path, with or without a minimum vertical separation distance. By topmost placement we mean the placement that makes the new lower envelope as high as possible. Again, we may fail to find such a path if a previously placed path will be intersected by the new path, or the desired separation distance cannot be attained. If the new path can be placed, it will replace the pieces of the lower envelope we collected, except, possibly, for the outermost two. These may be truncated horizontally. In the tree $\mathcal{T}$, this comes down to deleting the leaves in between μ and μ'. Up to three new leaves are inserted instead. Appropriate updates have to be done at μ and μ'.

In this concrete case with {HDH, VDV}-paths that are L_2-shortest, the end-

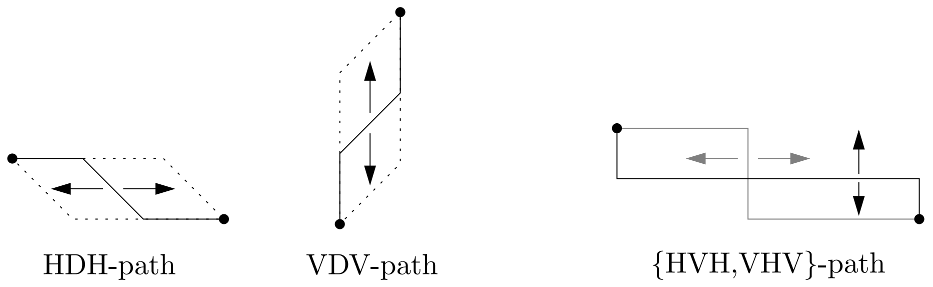


Figure 5.7: Some types of schematic paths.

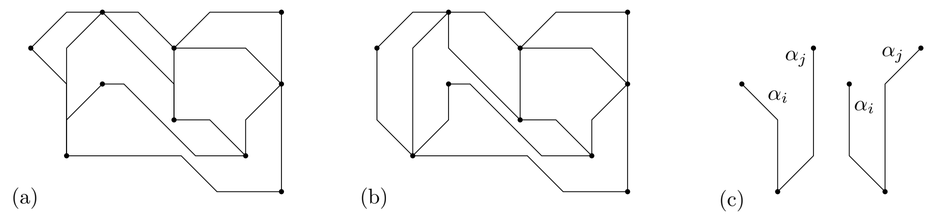


Figure 5.8: (a) A schematized map with 3-link, {HDH, VDV}-paths with shared departure from common endpoints; topmost placement. (b) Same, but without shared departure. (c) The indecision in aboveness for α_i and α_j when shared departure is not allowed.

points of the next path to be placed provides us a decision between a HDH-path or a VDV-path because the path has to be L_2-shortest: we need a HDH-path when the vertical distance between the endpoints is smaller than the horizontal distance, and otherwise we need a VDV-path. In general, this property does not hold for certain combinations of types of schematic paths, for example {HVH, VHV}-paths that are L_2-shortest (Figure 5.7). In this case, whether we place a HVH-path or a VHV-path can influence whether we can place another, later schematic path or not. To understand what types of schematic paths our method can handle, we make the following definitions.

Definition 5.12 *A* schematic map model *specifies, for any two points p, q in the plane, the collection of (schematic) paths with p and q as endpoints that can be used. A schematic map model is x-*monotone *if all paths in any collection that it specifies are x-monotone paths. A schematic map model is* ordered *if, for any two points p, q, and any two paths α_1, α_2 in the collection specified by the model for p, q, no two points a, b exist such that a is above α_1 and below α_2, and b is below α_1 and above α_2.*

In Figure 5.7, one can see that a model that allows for shortest {HVH, VHV}-paths is not ordered: points a and b can be placed in the two distinct bounded regions between the paths. However, L_2-shortest {HDH, VDV}-paths result in an ordered x-monotone schematic map model. Our algorithm can handle any ordered x-monotone schematic map model. Examples are {VDH, HDV, VDV, HDH}-paths that are L_1-shortest, {VH, HV}-paths, and {HVH, VHV}-paths where the distance between the endpoints determines which of the two types is used (for instance, when the horizontal distance between the endpoints is larger than the vertical distance, use a HVH-path, otherwise a VHV-path). Paths may degenerate to fewer links by using zero-length segments.

Besides the types of schematic paths, we may also specify a minimum vertical separation distance between two paths that do not have any shared endpoint. Observe that horizontal distance cannot be taken into account because this does not relate to the ordering used among paths.

In most cases, we can specify that two schematic paths with the same endpoint (shared endpoint) may not depart in the same direction from that endpoint. However, in some cases, it makes a difference which schematized path is placed first; see Figure 5.8(c). In a degenerate sense, the paths are above and below each other, giving a cycle in the placement order. Therefore, our algorithm cannot forbid shared departure in the vertical upward direction in all cases. Disallowing shared departure in the non-vertical directions can be incorporated efficiently, however, because the above-below relations give the placement order. Shared departure in the vertical downward direction is automatically avoided if it is possible, because every schematic path is placed topmost.

Theorem 5.13 *Given a map M with complexity n, and an ordered x-monotone schematic map model, a minimum vertical separation distance specified, and optionally, shared non-vertical departure disallowed, we can compute in $O(n \log n)$ time a schematized map equivalent to M satisfying the separation distance and shared departure conditions as specified, or report that no such map exists.*

Proof: By Theorem 5.10, we can detect in $O(n \log n)$ time if M has an equivalent, monotone map. In the negative case, no schematization of M is possible in an x-monotone schematic map model, and we are done. Otherwise, we can compute a total order extending the partial order among the paths in M in $O(n \log n)$ time by Corollary 5.11. By Lemma 5.7 we have that any monotone map with this order among its paths is equivalent to M. In particular, a map constructed with top-to-bottom placement as detailed in this section is equivalent to M.

It remains to show that we can place top-to-bottom all the schematic paths in $O(n \log n)$ time. Recall that we use a binary search tree $\mathcal{T}$ to represent the lower envelope of the placed schematic paths. To insert a new path, we find the two leaves μ and μ' with the x-coordinates of the endpoints of the new path in $O(\log n)$ time. Assume that k leaves lie in between μ and μ' in $\mathcal{T}$. Then we can determine in $O(k)$ time whether a placement of the new path exists, in any ordered x-monotone schematic map model, and with the separation distance and shared departure conditions. If a placement meeting the conditions does not exist, we can stop and report failure. Otherwise, we determine the topmost placement within the same time bound, asymptotically. The updating of $\mathcal{T}$

involves the deletion of k leaves, the insertion of $O(1)$ leaves, and the updating of two leaves μ and μ'. This takes $O((k+1)\log n)$ time. Since in total $O(n)$ leaves are inserted, and any leaf can be deleted only once, it follows that the total running time of placing the schematic paths is $O(n \log n)$.

If, at some moment, the algorithm cannot place the next schematic path because we would violate some condition, like placement closer than the specified minimum separation distance, or the placement would introduce an intersection, then the schematic map as desired does not exist. This follows from the fact that we maintain the topmost lower envelope among all possible placements of the previously placed schematic paths, which is well-defined in an ordered x-monotone schematic map model. □

Note that we always need to start ensuring that our original map admits an equivalent, monotone map, like we have done at the beginning of the previous proof. That is so because if we just compute the above-below relation among paths, for example by comparing each pair, and then place the schematic paths in the way that was explained in this section, we may construct a schematic map that is not equivalent to the original one, although it has the same above-below relation as the original map. This happens, for example, in Figure 5.4 right.

5.4 Experimental results

In order to test the quality of the output when applying the aforementioned ideas, a prototype has been implemented as a Java applet and can be seen on the web [2]. Although some of the techniques to speed up the algorithm that were explained are not used, the time of computation does not appear to be problematic.

Some editing tools were included in the prototype in order to make it more flexible and interactive for the user. Recall that the algorithm described above does not move the endpoints of paths, but some adjustments may be necessary. For example, it appeared to be useful to identify and merge the endpoints of paths when they are very close. Observe that our algorithm would report failure in the cases that the paths have a slight overlap in the vertical direction, or they are closer in the vertical direction than allowed; see Figure 5.9. When some path cannot be schematized, the prototype draws the original path. Also, the presence of intersections close to endpoints plays an important role, because this prevents an ordering of the paths. Sometimes, this type of problem, which can be present in inaccurate data, can be resolved by editing; see Figure 5.9.

From an aesthetic point of view, several practical improvements can be done during the placing of paths. For example, given the minimum vertical distance d to be kept between the schematic paths, this applet attempts to find the maximum distance $d' \geq d$ that permits the schematization of as many paths as d does. This improvement is done by a binary search over the integers (distance in pixels) to find this distance d'. Other improvements are to place paths in top-to-bottom and bottom-to-top order simultaneously, where the bottommost paths are schematized in the lowest placement possible. This adaptation yields

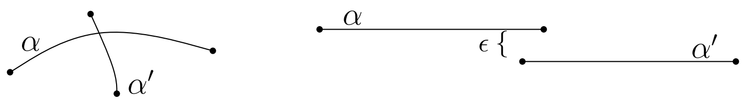


Figure 5.9: Some editing of the original map is necessary if it is topologically not correct or it has inaccuracies. Left: there is no order among paths. Right: the minimum separation distance between two paths cannot be respected.

maps that are more "rounded" than when all schematized paths are placed topmost. There are many other improvements that could be implemented too, like replacing corners by circular arcs to get C^1 continuity, parallel departure from endpoints represented by rectangles or bars, and so on.

The prototype was tested on maps of Canada, Ireland and Spain. They consist of between 769 and 1612 segments, and the time of computation is less than 3 seconds on a standard PC. The most time consuming part is the binary search to find the optimal distance, as explained at the end of previous paragraph. In all examples and results in this section, the smallest allowed vertical distance between two schematized paths was set to 10 pixels. Some of the results can be seen in Figures 5.10–5.12, where paths that could not be schematized are shown in their original shape and with thicker line style.

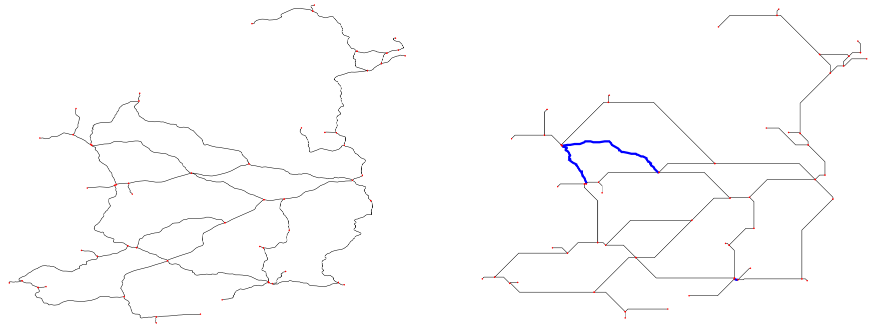


Figure 5.10: Ireland. Left: original railway map. Right: schematic map with {HDH, VDV}-paths and without shared departure.

For the maps of Ireland and Canada, a top-to-bottom placement order exists and hence, could be computed without any editing. But the map of Spain does not have a placement order. This is due to inaccuracies in the data, which is topologically not correct. Different paths had intersections that were not endpoints. After a few modifications with the editing tool, it was possible to

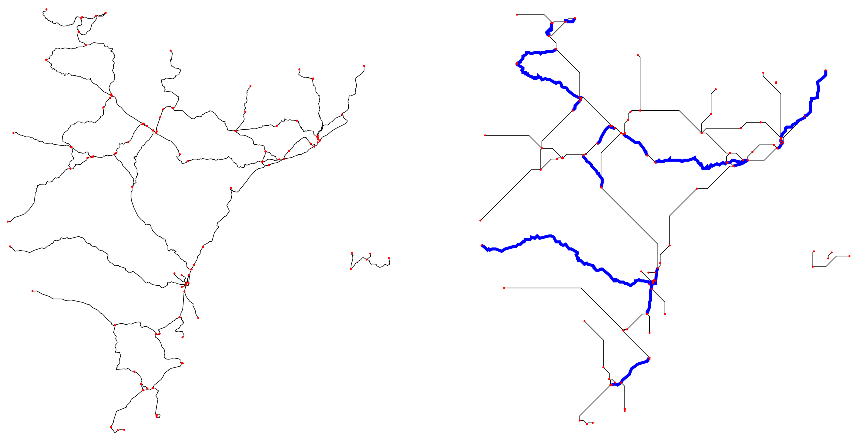


Figure 5.11: East of Spain. Left: original railway map. Right: schematic map with {HDH, VDV}-paths and without shared departure.

compute a placement order.

As can be seen in the pictures, some schematic paths cannot be placed because they would intersect some other one already placed, give shared departure, or would not respect the specified distance. It appears in the figures that more schematic paths could be placed, like the two non-schematized paths in Figure 5.10. However, there are two endpoints very close together at the upper end of the non-schematized paths, which would yield a placement too close to another path with different endpoints. Editing of the original map would be necessary to resolve this. In Table 5.1 we can see how many paths were present originally, and which percentage of these couldn't be placed. After analyzing the pictures in the prototype, which allows us to zoom in the original map, one can see that the biggest source of problems in all maps is indeed that some endpoints are very close together.

Figure	Total number of paths	A	B
5.10 right	65	61	61
5.11 right	100	76	79
5.12 center	155	115	134
5.12 bottom	155	139	139

Table 5.1: Data for the maps shown in the figures. A: number of schematic paths when shared departure is not allowed. B: number of schematic paths when shared departure is allowed.

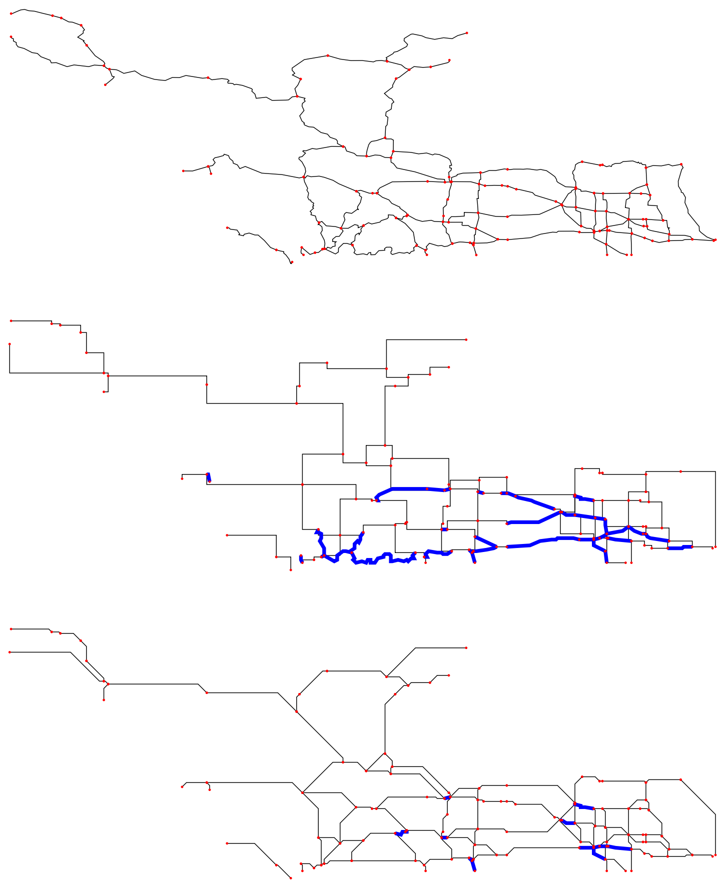


Figure 5.12: West of Canada. Top: original road map. Center: schematic map with two links per path without shared departure. Bottom: schematic map with {HDH, VDV}-paths without shared departure.

5.5 Concluding remarks

This chapter describes an efficient approach to compute schematized maps. Contrary to previous methods, our algorithm uses a bounded number of links per path, each of a given orientation (horizontal, vertical, or diagonal), as commonly used in transportation maps. Our approach, in contrast to previous works, is not an iterative, hopefully converging process, but it uses a combinatorial approach and always gives a correct solution if one exists. Experimental results that show the quality of the output have been presented.

There are many directions for further research. Firstly, we would like to be able to disallow shared departure for paths leaving the same endpoint in all cases, without sacrificing efficiency. This appears to be difficult for the vertical upward direction, if we want to guarantee a solution when one exists. Secondly, there are several types of schematic paths that cannot be handled by our method. It would be interesting to develop efficient algorithms that find a result if one exists in these cases too. Finally, before a schematization algorithm like ours can be useful in practice, certain local improvements and display styles must be incorporated, and more experimental work in order to improve the aesthetic quality of the output has to be done. Depending on the requirements for an aesthetic schematic map that follows all cartographic rules, this may involve more complex versions of the model than considered in this chapter.

Chapter 6

Linear cartograms

In this chapter, we consider the problem of finding a planar embedding of a (planar) graph with a prescribed Euclidean length on every edge. There has been substantial previous work on the problem without the planarity restrictions, which has close connections to rigidity theory. For general graphs, a reconstruction is always unique and easy-to-compute for a complete graph of (exact) distances, or any graph that can be "shelled" by incrementally locating nodes according to the distances to three noncollinear located neighbors (Figure 6.1). More interesting is that such graphs include visibility graphs [41] and segment visibility graphs [69]. In general, however, the reconstruction problem is NP-hard [122], even in the strong sense [115]. The uniqueness of a reconstruction in the generic case (in 2D) was recently shown to be testable in polynomial time by a simple characterization related to generic rigidity [83, 91], but this result has not yet led to efficient algorithms for actual reconstruction in the generic case.

Motivated by our application into linear cartograms, as discussed in Chapter 1, we consider a variation on this basic problem of reconstruction from distances: given a planar graph with prescribed lengths on the edges, construct a planar embedding of the graph that adheres to the specified edge lengths, and determine whether this embedding is unique, or determine that no such embedding exists.

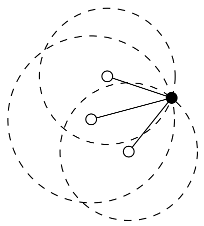


Figure 6.1: Locating a vertex from the distances to three located neighbors.

In the rest of the chapter, we show the following main results:

1. For planar 3-connected graphs, we can decide in $O(|V|)$ time whether there is a planar embedding with specified edge lengths in which only the outer face is not a triangle. Furthermore, such an embedding is always unique up to rigid motions (translations, rotations, and reflections), and can be constructed in $O(|V|)$ time. More generally, we can find planar embeddings in which the triangular faces form a connected family of cells and the nontriangular faces form a forest of cells. This extends the results by Di Battista and Vismara [52], where under the assumption that the graph is triangulated and the outer face is convex, the authors can test embeddability in linear time, but without providing an actual embedding. See Section 6.1.

2. Even for planar 3-connected graphs, deciding planar embeddability with unit edge lengths is strongly NP-hard, even when the embeddings are guaranteed to be infinitesimally rigid.[1] This improves upon results by Whitesides [120], where weak hardness was shown for (2-connected) planar linkages, and upon the work by Eades and Wormald [56], where the strong hardness is shown for 2-connected graphs with unit edge lengths and for 3-connected graphs with arbitrary edge lengths. Another (aesthetic) difference with respect to [56] is that our reduction is directly from planar 3-SAT, rather than using a synthetic problem as a bridge. See Section 6.2.

These results give a fairly precise division between tractable and intractable forms of planar embedding with specified edge lengths. Triangles seem to play a more fundamental role than other rigid structures, despite the close connections between rigidity and embedding with specified edge lengths [40, 83, 91]. Other than visibility graphs [41, 69] and dense graphs [19], our results are the first positive results for efficient embeddings of (special) graphs with specified edge lengths.

6.1 Triangulated graphs

Let G be a 3-connected planar graph. By Whitney's Theorem, G has only one topological embedding into the 2-dimensional sphere, or equivalently the faces in any planar embedding of G are always induced by the same cycles [53, Chapter 6]. In particular, all embeddings of G have the same dual graph G^*, and once we have fixed the outer face, the topological embedding into the plane is completely determined. This is the basic ingredient for the following result:

Theorem 6.1 *If $G = (V, E)$ is a 3-connected graph with specified edge lengths, we can decide in $O(|V|)$ time on a real RAM whether there is a planar embedding such that all faces are triangles, with the possible exception of the outer face.*

[1]Infinitesimal rigidity is a strong form of rigidity, stating that no first-order motion of the vertices preserves the lengths of the edges to the first order. See e.g. [79] for formal definitions.

Proof: Consider any planar embedding of G, which can be computed in $O(|V|)$ time [51]. If two or more faces are not triangles, then we can decide that the desired realization is not possible because of Whitney's theorem. If exactly one face is not a triangle, that face must be the outer face in the desired realization. If all faces are triangles, any longest edge has to be part of the outer face, which gives us at most two candidates T and T' for the outer face. If T is the outer face, then T' must fit inside T while sharing the common edge, and vice versa. This test leaves us with at most one candidate for the outer face f_{ext}.

All nodes in $G^* \backslash f_{ext}$ correspond to triangular faces. We pick a node f_0 in this graph, and compute coordinates for the vertices of the corresponding triangle that realize the triangle edge lengths. Now we visit all nodes in $G^* \setminus f_{ext}$ using breadth-first search from f_0. When visiting a node f_i, two options arise:

1. If all vertices of the face f_i have already been assigned coordinates, we check that all the edges in f_i have the specified edge lengths.

2. If some vertex of the face f_i has not been assigned coordinates, we know that the other two vertices u, v of f_i participate in another face f_j that has been already visited, and so they have already been assigned coordinates. We can compute the coordinates of the third vertex using the specified edge lengths and the restriction that f_i and f_j must lie on opposite sides of the line segment uv due to Whitney's Theorem.

At the end, every edge in the graph has been checked whether it satisfies the specified edge length, including the lengths of the edges of the outer face f_{ext}. In the process, we visited each face once, and we spent constant time per face, so, overall, the embedding process takes $O(|V|)$ time.

We need to check that the realization that we constructed is indeed planar, to avoid situations like the ones depicted in Figure 6.2. A simple plane sweep would do this in $O(|V| \log |V|)$ time. To get linear time, we first construct a triangulation of the whole plane: We enclose all points in a large triangle T and triangulate the area between T and the boundary of the outer face f_{ext}. To do this, we insert an edge from an extreme vertex of V to a corner of T and triangulate the resulting simple polygon in linear time [39]. Under the assumption that the original embedding was planar, we obtain a graph which is a triangulation of T and is embedded in the plane without crossings. On the other hand, if the original embedding contains crossings, the triangulation algorithm will either (i) terminate in error, or (ii) it will produce a subdivision of T which is topologically consistent but whose embedding contains crossings. Topological consistency means that the two triangle faces incident to an edge are embedded on different sides of the edge, except for the edges of T where the other triangle is embedded inside T. The existence of crossings (ii) for a convex subdivision can be tested in linear time [48]. $\square$

Observe that, in the proof of the Theorem 6.1, we have only used that the coordinates of the vertices can be computed by considering the triangular faces in an appropriate order. To get this property, we only need that each vertex of G

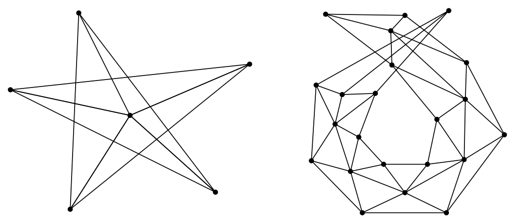


Figure 6.2: These examples show that we need to check that the embedding is indeed planar.

is incident to a triangular face, and that the set of triangular faces is connected in the dual graph G^*. Once we have fixed the outer face, these hypotheses would be enough to prove the result.

This result extends the one by Di Battista and Vismara [52] in two ways. Firstly, we do not need to assume that the outer face is convex, although in return we have to use linear time triangulation [39]. To avoid using the result in [39], a checker for non-convex subdivisions should be developed, but this problem remains elusive [22, Problem 21]. Secondly, we do not need to assume that the outer face is fixed, which becomes relevant when all the faces of the graph are triangles.

In the previous theorem, we have assumed a real RAM model when computing the coordinates. This model is customary in computational geometry; see [110] for a description. In the Turing machine model, if *all* faces are triangles we can *test* embeddability, without constructing coordinates, as follows. We start detecting the outer face T in linear time like we did in the previous proof, and then use the result in [52]: the graph G admits a planar realization if and only if, for all vertices $v \notin T$, the sum of the angles that are incident to v is 360 degrees.

The cosine of an angle incident to v can be described by an algebraic expression on the edge lengths incident to it because of the cosine law. The cosine and sine of a sum of angles can be expressed as a polynomial on the cosines and sines of the angles. Therefore, the condition that the sum of the angles that are incident to v is 360 degrees can be reduced to an evaluation of polynomials: starting from an angle, we consider the sine and cosine of the clockwise partial sums of angles. Studying their signs, we can make sure that the partial sums do not exceed 360 degrees, and then the sum of the angles is 360 degrees if an only if the cosine and the sine of the sum are 1 and 0, respectively.

The maximum degree of the polynomials that we evaluate depends on the degree of the vertices. For graphs of bounded degree, the polynomials have bounded degree, and the condition can be tested in polynomial time in the classical Turing machine model, with rational edge lengths as inputs, using separation bounds for algebraic computations, see [25, 26, 94]. We may also allow square roots of rationals as inputs. (Otherwise, it will be difficult to come up with interesting examples of realizable graphs with rational edge lengths.)

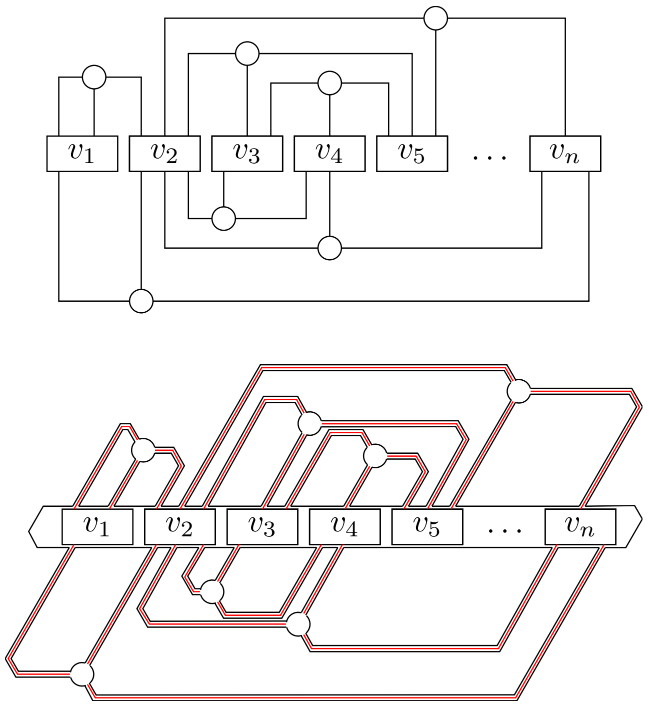


Figure 6.3: Top: example of a planar 3-satisfiability instance. The variables can be arranged on a straight line, and the clauses are represented as a vertex with three orthogonal edges leaving from it and one bend in each edge. Bottom: High-level sketch of NP-hardness reduction. Each line will be replaced by a rigid 3-connected structure.

For general graphs, this algorithm is singly-exponential in the degree.

6.2 NP-hardness

To show the NP-hardness of deciding if a planar embedding with specified edge lengths exist, we reduce from the P3-SAT (planar 3-satisfiability) problem, which is strongly NP-complete [95]. In an instance of P3-SAT, we are given a planar bipartite graph whose nodes on one side of the bipartition represent the variables $v_1, \ldots, v_n$, and whose nodes on the other side represent the clauses $C_1, \ldots, C_m$, and edges connect each clause to the three variables it contains. Moreover, the variables can be arranged on a horizontal line, and the three-legged clauses be drawn such that all edges lie either above or below this line; and the graph can be drawn on a rectangular grid of polynomial size as shown in Figure 6.3, top [93].

The high-level workings of the reduction are as follows. We slant the grid into a hexagonal grid to get angles that are multiples of 60 degrees. This slant

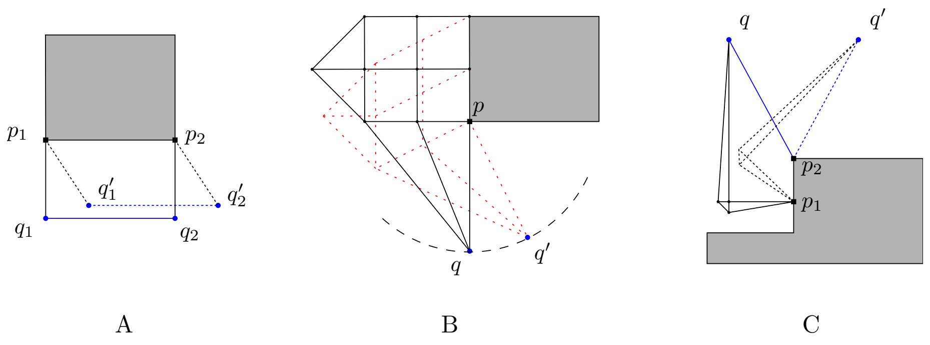


Figure 6.4: Assume that the grey regions are rigid and fixed. A. The segments p_1p_2 and q_1q_2 are parallel in any realization. B. How to make rotations while keeping 3-connectedness. C. The vertex q can only be realized in two positions.

will allow us to make all lengths one. Furthermore, we modify the drawing so that all the corners have angles of 120 degrees, and the three edges arriving at a clause form angles of 120 degrees; see Figure 6.3, bottom. We make a rigid structure that will leave a tunnel for each edge connecting a variable with a clause. A variable will be represented by a rigid structure that has two different realizations, representing the truth assignment of the variable. The value of the literal will be transmitted to the clause through the tunnel corresponding to the edge, and we will represent the clause by a structure that can be realized if and only if at least one of the literals is true. Furthermore, each of the lines in the figure will be represented by a rigid 3-connected bar, like a "thick" line. This will be the basic trick to make the whole graph 3-connected as well.

The construction relies on three basic rigid structures that are depicted in Figure 6.4, and that we explain in the following. In all cases, the grey regions represent 3-connected, rigid structures which are fixed. Firstly, in Figure 6.4A, the edges p_1q_1 and p_2q_2 have the same length, and so do p_1p_2 and q_1q_2. Under these conditions, in any realization of this structure, the edges p_1p_2 and q_1q_2 have to be parallel. Secondly, in Figure 6.4B, there is a 3-connected structure that allows q to rotate around p. Finally, in Figure 6.4C, if the vertices p_1 and p_2, marked with squares, are fixed, then the vertex marked with a circle has two possible positions, q and q'. This is so because the distance between this vertex and p_1 and p_2 is fixed, and therefore it has to be placed at the intersection of two circles centered at p_1 and p_2.

Theorem 6.2 *Deciding planar embeddability of a planar 3-connected graph with unit edge lengths is NP-hard.*

Proof: We have already described the general idea, so it only remains to describe the gadgets that are used. For the tunnels, we need a structure that allows us to fix the relative positions of both sides of the tunnel, while transmitting the value of the literal through the tunnel. The value will be either true or false, so we need a structure that allows two realizations.

In Figure 6.5A the *holder* gadget is shown. Consider the upper half of it. Observe that the two points that are marked with big dots, p_1, p_2, and the two points that are marked with squares, q_1, q_2, represent a situation like shown in Figure 6.4A. Therefore, the bar that supports q_1, q_2 is always parallel to the one that supports p_1, p_2, and the point q_2 is always vertically above point q. The points q, q_2 and p_2, implement the idea shown in Figure 6.4C, and so p_2 has only two possible placements with respect to q, q_2. Overall, this implies that the upper half of Figure 6.5A can be realized in two ways. The lower half of the holder is a mirrored copy of upper half, and so it also has two realizations.

The holder gadget can be realized in four different ways: two of them keep the relative position of both sides of the tunnel (Figure 6.5A and B), while two of them would move them (Figure 6.6A and B). We can concatenate two of these gadgets with one *bend*, as shown in Figure 6.6C, in such a way that the realizations in Figure 6.6A and B are not possible. Thus, the two sides of the tunnel are connected in a (globally) rigid way. We define the *transmitter* to be the bar that is inside the tunnel, because it will transmit the truth value of the literal from the variable to the clause. Observe that in one of the possible realizations of the holder, the transmitter is shifted four units with respect to the other possible realization. Below we will discuss the meaning of the possible realizations of the transmitter.

The structure that we have described is 3-connected, and so we can construct a rigid 3-connected structure, as shown in Figure 6.3, bottom, where the distance between the upper and the lower part will be defined later on by the height of the variables. The sides of the tunnels taken together form a rigid structure in which the transmitters and the variables can move: If a tunnel contains a bend, its two sides can be connected rigidly by two holders, as in Figure 6.6C. One can check that the sides of a tunnel without a bend are always connected to a tunnel with a bend, and therefore are also immobile. (Or we could introduce two bends in a zigzag way to make an otherwise straight tunnel rigid in its own right.)

We still have to discuss how the variables, the transmitter, and the clauses work.

For each variable we repeat the structure of the upper half of the holder gadget, but with a thicker bar (*variable-bar*) inside; see Figure 6.7. Consider the realization of the structure assigned to true. On the sides of the variable-bar that are facing the tunnels, for each literal that is not negated, we place an indentation on it that prolongates the tunnel of the literal. For the literals that are negated, we place such an indentation on the part of the variable bar that faces the tunnel in the "false" realization of the structure; see Figure 6.7. We have to make the variable-bar large enough that tunnels for all occurrences of each variable can be accommodated on its sides. (In Figure 6.7, there are three tunnels on each side.)

The graph that we have constructed so far is 3-connected and rigid. Furthermore, whenever a literal is true, the transmitter bar inside the tunnel can be pushed towards the variable-bar. Furthermore, we can transmit this "pushing information", or pressure, through the tunnel, and also through the corners

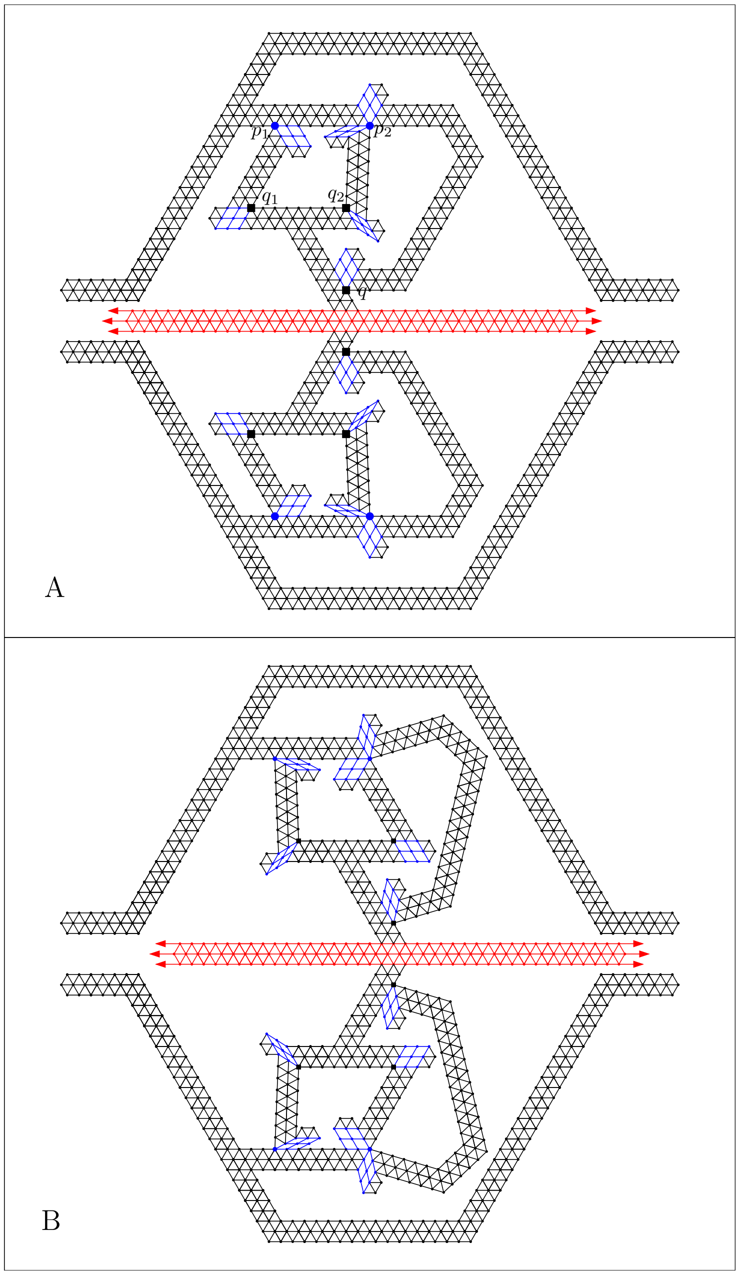


Figure 6.5: A. The holder gadget. A–B. Two possible realizations of the holder. In B, the transmitter is four units to the right with respect to its position in A.

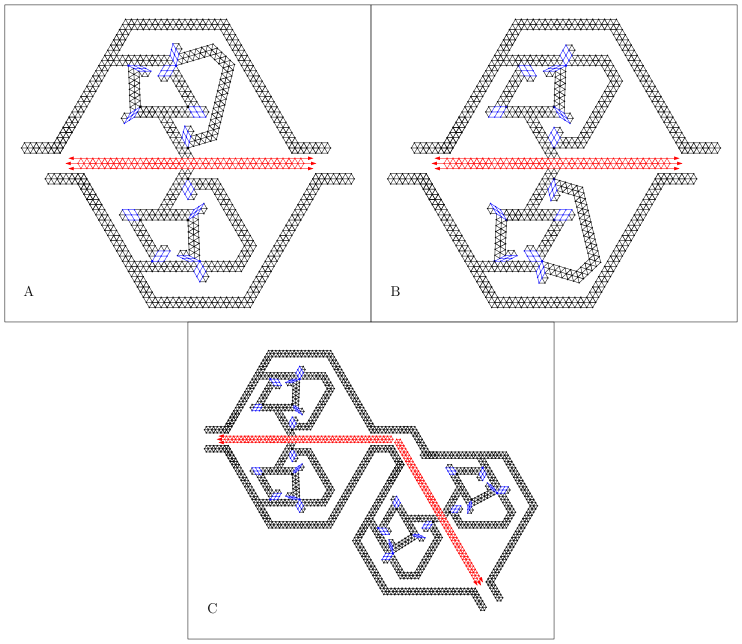


Figure 6.6: A–B. Two other possible realizations of the holder. C. We avoid these realizations connecting by a bend two consecutive holders. This rigidly connects the sides of the tunnel.

using Figure 6.6C, so that it can be used at the clause.

Our next goal is to design a *clause checker* that is realizable if and only if one of the three transmitters can be pushed towards its variable. It turns out to be easier to solve the reverse problem: a clause that is realizable if and only if one of the transmitters is pushed towards the clause. Therefore, we design a pushing-inverter which we place on each tunnel just before the clause. It is described in Figure 6.8, where its two possible realizations are displayed. In particular, the top two thick dots correspond to the possible positions of a vertex depending on whether the pressure is towards the clause or the variable. The *inverter* gadget changes pressure towards the clause into pressure towards the variable, and vice versa. We can make it 3-connected by putting a holder gadget just before it, and another holder gadget immediately after it.

Finally, a clause is described in Figure 6.9, with its relevant realizations. The big dots in each literal are at four units distance, and they indicate the two possible positions for the end of the transmitter. The one that is closer to the center indicates that the literal is true (pushing towards the clause). In all cases, the position of the big dot in the center is completely determined by the values of l_i and l_j. When all l_i, l_j, l_k are false, then the big dot in the center is too far from l_k to be realizable; see Figure 6.9A. In the other cases, it is always realizable; see Figure 6.9B–D for some cases.

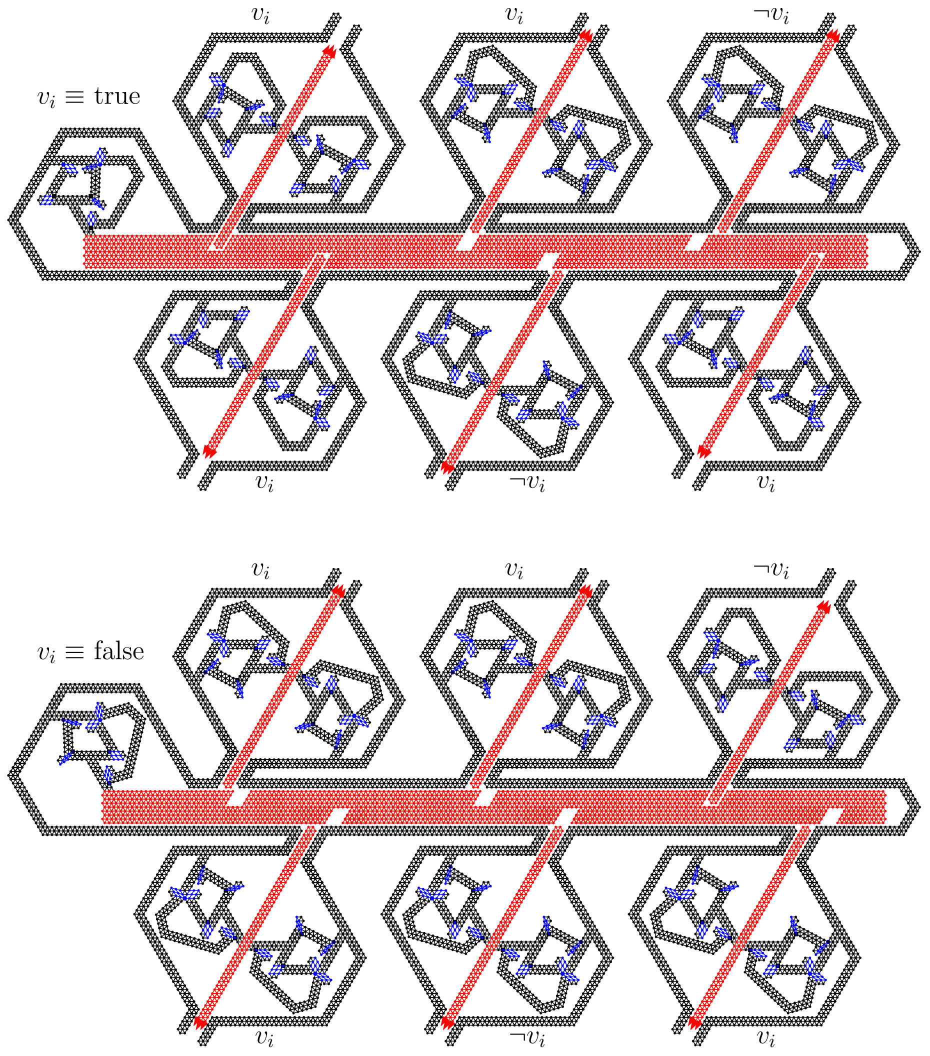


Figure 6.7: A variable assigned to true (top) and false (bottom). The transmitter can be pushed towards the variable only when the literal is true. Observe that in this case, if some other literal in the same clause is true, then the transmitter does not need to come into the indentation.

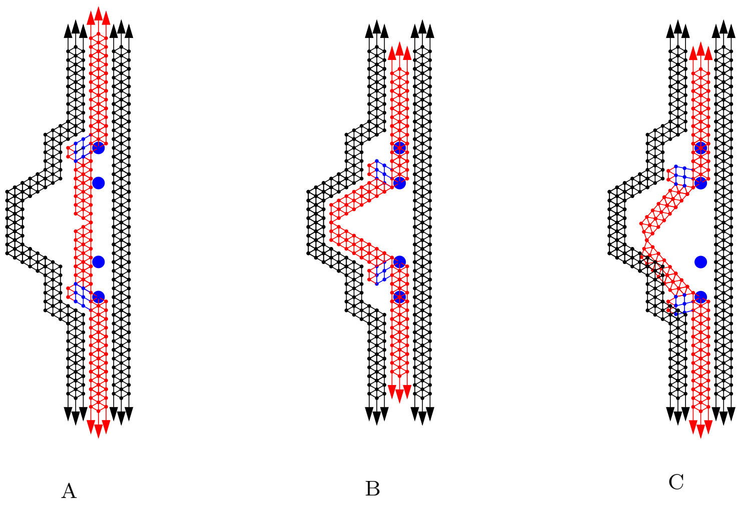


Figure 6.8: An inverter. A–B are realizable, but C is not.

To conclude, we summarize the argument why a realization of the graph corresponds to a satisfying truth assignment. The clause checker can be realized if and only if at least one transmitter is at the position closer to the clause checker. This can only be the case if, at the variable side of the corresponding inverter, the transmitter is pushed away from the clause checker. This pushing is transmitted through all bends and holders to the variable wheels. It follows that the literal must be true.

One can check by inspection that the clause-gadget is 3-connected, and therefore the whole construction is 3-connected. Furthermore, the lengths of the constructed graph are one because the vertices and edges lie on a hexagonal grid. The grid has polynomial size. Therefore, we are using a polynomial number of edges and bits, and so the reduction can be done in polynomial time. □

Our NP-hardness result is in the standard Turing machine model because we construct a graph with a polynomial number of edges, all of them with unit edge length. On the other hand, it is not clear that the problem belongs to NP, as the coordinates of the embedding are not rational or algebraic numbers of bounded degree.

The 3-SAT problem is NP-hard even if each variable occurs at most 6 times, and this property is maintained in the reduction from 3-SAT to P3-SAT [95]. If a variable needs to accommodate at most six tunnels, then a variable is formed by a bounded number of edges, the faces that participate in the variable gadget have bounded degree. By filling the free space between the tunnels and on the outside by a triangulation, we can make sure that all the faces have bounded degree. Therefore, the problem remains NP-hard even if we assume bounded face degree.

Observe that when the graph is realizable, the realization is infinitesimally

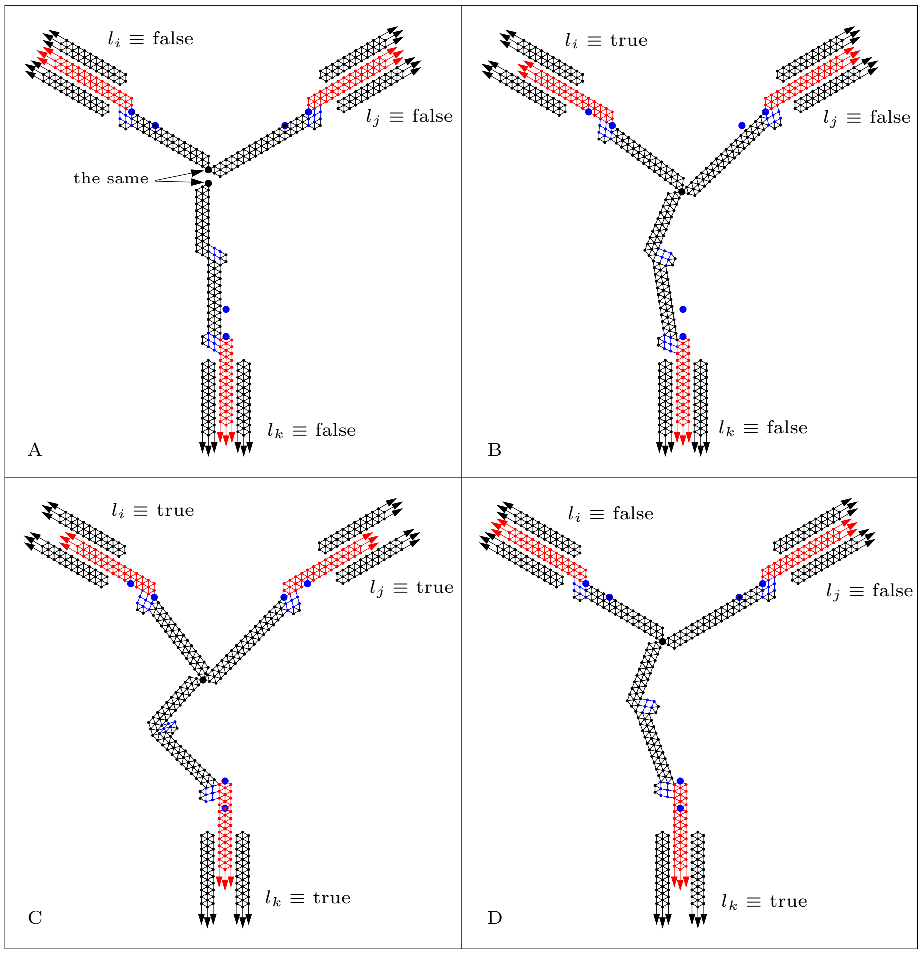


Figure 6.9: A clause checker. The situation in A is not realizable, but the ones in B–D are realizable.

rigid. In other words, its vertices cannot be infinitesimally perturbed in a way that preserves the edge lengths to the first order. This condition is stronger than rigidity, and implies that the underlying graph is generically rigid [79]. Therefore, the problem remains NP-hard even when we know that the graph is generically rigid.

Bibliography

[1] Webpage: `http://www.esri.com/software/arcgis/arcgisxtensions/schematics/index.html`.

[2] `http://www.cs.uu.nl/archive/sw/schematic-map/`.

[3] M. Abellanas, F. Hurtado, and P.A. Ramos. Structural tolerance and Delaunay triangulation. *Information Processing Letters*, 71:221–227, 1999.

[4] P. K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29:912–953, 1999.

[5] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

[6] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30:412–458, 1998.

[7] M. Agrawala and C. Stolte. Rendering effective route maps: Improving usability through generalization. In *Proc. SIGGRAPH 2001*, pages 241–250, 2001. `http://graphics.stanford.edu/papers/routemaps/`.

[8] R. Aharoni and P. Haxell. Hall's theorem for hypergraphs. *Journal of Graph Theory*, 35:83–88, 2000.

[9] M. A. Armstrong. *Basic Topology.* McGraw-Hill, London, UK, 1979.

[10] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey, 2002. `http://www.cs.princeton.edu/~arora/pubs/arorageo.ps`.

[11] S. Avelar. *Schematic Maps on Demand: Design, Modeling and Visualization.* PhD thesis, Swiss Federal Institute of Technology, Zurich, 2002.

[12] S. Avelar and R. Huber. Modeling a public transport network for generation of schematic maps and location queries. In *Proc. 20th Int. Cartographic Conference*, pages 1472–1480, 2001.

[13] S. Avelar and M. Müller. Generating topologically correct schematic maps. In *Proc. 9th Int. Symp. on Spatial Data Handling*, pages 4a.28–4a.35, 2000.

[14] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41:153–180, 1994.

[15] T. Barbowsky, L.J. Latecki, and K. Richter. Schematizing maps: Simplification of geographic shape by discrete curve evolution. In *Spatial Cognition II, LNAI 1849*, pages 41–48, 2000.

[16] C. Baur and S.P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30:451–470, 2001. A preliminary version appeared in *APPROX'98*.

[17] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[18] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[19] B. Berger, J. Kleinberg, and T. Leighton. Reconstructing a three-dimensional model with arbitrary errors. In *Proc. 28th Annu. ACM Sympos. Theory Comput.*, pages 449–458, May 1996.

[20] M. W. Bern and D. Eppstein. Approximation algorithms for geometric problems. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 8, pages 296–345. PWS Publishing, 1996.

[21] S. Bespamyatnikh. Computing homotopic shortest paths in the plane. In *SODA03*, pages 609–617, 2003. To appear in J. of Algorithms.

[22] F. Brandenberg, D. Eppstein, M.T. Goodrich, S.G. Kobourov, G. Liotta, and P. Mutzel. Selected open problems in graph drawing. In *Graph Drawing (Proc. GD'03)*, LNCS, 2003. To appear.

[23] U. Brandes, G. Shubina, R. Tamassia, and D. Wagner. Fast layout methods for timetable graphs. In J. Marks, editor, *Graph Drawing, Proceedings of 8th International Symposium, GD 2000*, volume 1984 of *Lecture Notes in Computer Science*, pages 127–138. Springer-Verlag, 2001.

[24] U. Brandes and D. Wagner. Using graph layout to visualize train interconnection data. *Journal of Graph Algorithms and Applications*, 4(3):135–155, 2000. `http://www.cs.brown.edu/publications/jgaa/volume04.html`.

[25] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27(1):87–99, 2000.

[26] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. In F. Meyer auf der Heide, editor, *Algorithms — ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28–31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 254–265. Springer-Verlag, 2001.

[27] B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1–26, 1985.

[28] S. Cabello. Approximation algorithms for spreading points. Technical report UU-CS-2003-040, Available at `http://www.cs.uu.nl/research/techreps/UU-CS-2003-040.html`, 2003.

[29] S. Cabello, M. de Berg, S. van Dijk, M. van Kreveld, and T. Strijk. Schematization of road networks. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 33–39, 2001.

[30] S. Cabello, M. de Berg, and M. van Kreveld. Schematization of networks. Technical report, University Utrecht. Submitted to journal, 2002.

[31] S. Cabello, E.D. Demaine, and G. Rote. Planar embeddings of graphs with specified edge lengths. In *Graph Drawing 2003*, LNCS, 2004. To appear.

[32] S. Cabello, Y. Leo, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. *Discrete & Computational Geometry.* To appear. A preliminary version appeared in *SoCG'02*.

[33] S. Cabello and M. van Kreveld. Schematic networks: an algorithm and its implementation. In D.E. Richardson and P. van Oosterom, editors, *Advances in Spatial Data Handling*, pages 475–486. Springer, 2002.

[34] S. Cabello and M. van Kreveld. Approximation algorithms for aligning points. *Algorithmica*, 37:211–232, 2003. A preliminary version appeared in *SoCG'03*.

[35] J. Campbell. *Map Use and Analysis.* McGraw-Hill, Boston, 4th edition, 2001.

[36] S. Čapkun, M. Hamdi, and J. Hubaux. GPS-free positioning in mobile ad-hoc networks. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, pages 3481–3490, January 2001.

[37] B. Chandra and M. M. Halldórsson. Approximation algorithms for dispersion problems. *J. Algorithms*, 38:438–465, 2001.

[38] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

[39] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[40] R. Connelly. On generic global rigidity. In P. Gritzman and B. Sturmfels, editors, *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, volume 4 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 147–155. AMS Press, 1991.

[41] C. Coullard and A. Lubiw. Distance visibility graphs. *Internat. J. Comput. Geom. Appl.*, 2(4):349–362, 1992.

[42] G. M. Crippen and T. F. Havel. *Distance Geometry and Molecular Conformation.* John Wiley & Sons, 1988.

[43] F.H. Croom. *Basic Concepts of Algebraic Topology.* Springer Verlag, Berlin, 1978.

[44] M. de Berg and O. Schwarzkopf. Cuttings and applications. *Internat. J. Comput. Geom. Appl.*, 5:343–355, 1995.

[45] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

[46] M. de Berg, M. van Kreveld, and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and GIS*, 25:243–257, 1998.

[47] B.D. Dent. *Cartography: Thematic Map Design.* McGraw-Hill, 5th edition, 1999.

[48] O. Devillers, G. Liotta, F. P. Preparata, and R. Tamassia. Checking the convexity of polytopes and the planarity of subdivisions. *Comput. Geom. Theory Appl.*, 11:187–208, 1998.

[49] T. K. Dey and S. Guha. Transforming curves on surfaces. *Journal of Computer and System Sciences*, 58:297–325, 1999.

[50] T. K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete Comput. Geom.*, 14:93–110, 1995.

[51] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.

[52] G. Di Battista and L. Vismara. Angles of planar triangular graphs. *SIAM Journal on Discrete Mathematics*, 9(3):349–359, 1996. A preliminary version appeared in *STOC'93.*

[53] R. Diestel. *Graph Theory.* Springer-Verlag, New York, 2nd edition, 2000.

[54] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.

[55] C.A. Duncan, A. Efrat, S.G. Kobourov, and C. Wenk. Drawing with fat edges. In *Graph Drawing 2001*, volume 2265 of *LNCS*, pages 162–177, 2002.

[56] P. Eades and N. Wormald. Fixed edge length graph drawing is NP-hard. *Discrete Appl. Math.*, 28:111–134, 1990.

[57] H. Edelsbrunner. A note on dynamic range searching. *Bull. EATCS*, 15:34–40, 1981.

[58] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[59] H. Edelsbrunner, Leonidas J. Guibas, J. Hershberger, R. Seidel, Micha Sharir, J. Snoeyink, and Emo Welzl. Implicitly representing arrangements of lines or segments. *Discrete Comput. Geom.*, 4:433–466, 1989.

[60] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[61] H. Edelsbrunner and E. Waupotitsch. A combinatorial approach to cartograms. *Comput. Geom. Theory Appl.*, 7:343–360, 1997.

[62] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[63] A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir. Dynamic data structures for fat objects and their applications. *Comput. Geom. Theory Appl.*, 15:215–227, 2000. A preliminary version appeared in *WADS'97, LNCS 1272*.

[64] A. Efrat, S.G. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. In *10th Annual European Symposium, ESA'02*, volume 2461 of *LNCS*, pages 411–423, 2002.

[65] D. Elroi. Designing a network line-map schematization software enhancement package. In *Proc. 8th Ann. ESRI User Conference*, 1988. `http://www.elroi.com/fr2_publications.html`.

[66] D. Elroi. GIS and schematic maps: A new symbiotic relationship. In *Proc. GIS/LIS'88*, 1988. `http://www.elroi.com/fr2_publications.html`.

[67] D. Elroi. Schematic views of networks: Why not have it all. In *Proc. of the 1991 GIS for Transportation Symposium*, pages 59–76, 1991. `http://www.elroi.com/fr2_publications.html`.

[68] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom.*, 16:389–418, 1996.

[69] H. Everett, C. T. Hoàng, K. Kilakos, and M. Noy. Distance segment visibility graphs. Manuscript, 1999. `http://www.loria.fr/~everett/publications/distance.html`.

[70] S.P. Fekete and H. Meijer. Maximum dispersion and geometric maximum weight cliques. To appear in *Algorithmica* 38(3), 2004.

[71] J. Fiala, J. Kratochvíl, and A. Proskurowski. Geometric systems of disjoint representatives. In *Graph Drawing, 10th GD'02, Irvine, California*, number 2528 in Lecture Notes in Computer Science, pages 110–117. Springer Verlag, 2002.

[72] J. Fiala, J. Kratochvíl, and A. Proskurowski. Systems of sets and their representatives. Technical Report 2002-573, KAM-DIMATIA, 2002. Available at `http://dimatia.mff.cuni.cz/`.

[73] H.N. Gabow. A matroid apporach to finding edge connectivity and packing arborescences. *J. Comput. Systems Sci.*, 50:259–273, 1995.

[74] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 392–402, 1988.

[75] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[76] S.H. Gerez. *Algorithms for VLSI Design Automation.* John Wiley & Sons, Chichester, 1999.

[77] A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-time parallel algorithms for matching and related problems. *Journal of Algorithms*, 14:180–213, 1993. A preliminary version appeared in *FOCS'88.*

[78] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *J. Algorithms*, 23:51–73, 1997.

[79] J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity.* American Mathematical Society, 1993.

[80] R. Grossi and E. Lodi. Simple planar graph partition into three forests. *Discrete Applied Mathematics*, 84:121–132, 1998.

[81] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48:798–859, 2001.

[82] A. Hatcher. *Algebraic Topology.* Cambridge University Press, 2001.

[83] B. Hendrickson. Conditions for unique graph realizations. *SIAM J. Comput.*, 21(1):65–84, February 1992.

[84] B. Hendrickson. The molecule problem: Exploiting structure in global optimization. *SIAM J. on Optimization*, 5:835–857, 1995.

[85] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.

[86] J. Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[87] I. Heywood, S. Cornelius, and S. Carver. *An Introduction to Geographical Information Systems.* Addison Wesley Longman, New York, 1998.

[88] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32:130–136, 1985.

[89] J. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[90] H. Imai and T. Asano. Efficient algorithms for geometric graph search problems. *SIAM J. Comput.*, 15(2):478–494, 1986.

[91] B. Jackson and T. Jordán. Connected rigidity matroids and unique realizations of graphs. Manuscript, March 2003.

[92] K. Kedem, R. Livne, J. Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[93] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. on Discrete Mathematics*, 5(3):422–427, August 1992.

[94] C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *Proc. 12th Annu. ACM–SIAM Sympos. Discrete Algorithms*, pages 496–505, 2001.

[95] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[96] F. M. Maley. *Single-Layer Wire Routing and Compaction.* MIT Press, Cambridge, MA, 1990.

[97] F.M. Maley. Testing homotopic routability under polygonal wiring rules. *Algorithmica*, 15:1–16, 1996.

[98] J. Matoušek. More on cutting arrangements and spanning trees with low crossing number. Technical Report B-90-2, Fachbereich Mathematik, Freie Universität Berlin, Berlin, 1990.

[99] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[100] J. Matousek. *Lectures on Discrete Geometry.* Springer Verlag, Berlin, 2002.

[101] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.

[102] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[103] M. Monmonier. *How to Lie with Maps.* The University of Chicago Press, Chicago, 1991.

[104] C. W. Mortensen. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 618–627. Society for Industrial and Applied Mathematics, 2003.

[105] C. W. Mortensen. Personal communication, 2003.

[106] J. R. Munkres. *Topology: A first course.* Prentice Hall, Englewood Cliffs, NJ, 1975.

[107] G. Neyer. Line simplication with restricted orientations. In *Algorithms and Data Structures, WADS'99*, volume 1663 of *LNCS*, pages 13–24, 1999.

[108] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 2nd edition, 1998.

[109] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56(4):304–311, 1994.

[110] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, October 1990.

[111] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proceedings of 6th Annual International Conference on Mobile Computing and Networking*, pages 32–43, Boston, MA, August 2000.

[112] R. Raghavan, J. Cohoon, and S. Sahni. Single bend wiring. *J. Algorithms*, 7:232–257, 1986.

[113] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[114] C. Savarese, J. Rabaey, and J. Beutel. Locationing in distributed ad-hoc wireless sensor networks. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 2037–2040, Salt Lake City, UT, May 2001.

[115] J. B. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. In *Proc. 17th Allerton Conf. Commun. Control Comput.*, pages 480–489, 1979.

[116] A. Schrijver. A course in combinatorial optimization. Lecture Notes. Available at `http://homepages.cwi.nl/~lex/files/dict.ps`, 2003.

[117] B. Simons. A fast algorithm for single processor scheduling. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 246–252, 1978.

[118] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[119] R. Weibel. Generalization of spatial data: Principles and selected algorithms. In *Lecture notes from CISM Advanced School on Algorithmic Foundations of Geographic Information Systems*, pages 99–152. Springer-Verlag, 1996.

[120] S. Whitesides. Algorithmic issues in the geometry of planar linkage movement. *Australian Computer Journal*, 24(2):42–50, May 1992.

[121] G. Woeginger. Personal communication, 2003.

[122] Y. Yemini. Some theoretical aspects of position-location problems. In *Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 1–8, 1979.

Curriculum Vitae

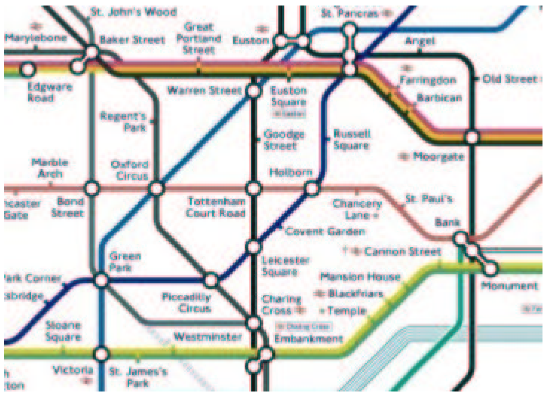
Sergio Cabello, born in 1977 in Lleida, Spain, got his degree (Llicenciat) in Mathematics in 1999 from the Universitat Politècnica de Catalunya, in Barcelona. In 2000 he started as PhD student (AIO) at at the Institute of Information and Computing Sciences of Utrecht University, where he completed this thesis in 2004.

Samenvatting

In een onbekende omgeving zijn kaarten een belangrijk hulpmiddel om te navigeren, en door onze reislust zien we ze steeds vaker. Kaarten met een bijzonder doel, bijvoorbeeld het toelichten van één route, zijn bijzonder nuttig. Gezien het toenemende gebruik van kaarten is het de moeite waard het ontwerpproces te automatiseren, zodat een plattegrond met dezelfde functionaliteit en kwaliteit met minder werk gemaakt kan worden. Dit is het doel van *geautomatiseerde kartografie*. Dit onderzoeksveld probeert met behulp van computers het werk van kartografen te verlichten.

Dit proefschrift onderzoekt geometrische vraagstukken die ontstaan bij het automatisch construeren van schematische netwerken: vereenvoudigde kaarten die veelal voor metro en trein worden gebruikt (zie de figuur). Bij deze kaarten wordt de echte kaart vervormd om zijn leesbaarheid, en dus bruikbaarheid, te vergroten. In dit proefschrift hebben we de volgende eigenschappen van een schematische kaart geanalyseerd:

- Verbindingen tussen stations moeten zo mogelijk horizontaal, verticaal of diagonaal zijn
- Als dat niet mogelijk is, dan moet de verbinding tussen twee stations



Detail van de metrokaart van Londen, een klassiek voorbeeld van een schematische kaart.

uit twee of drie rechte lijnstukken bestaan die horizontaal, verticaal of diagonaal moeten zijn.

- Drukke delen op de kaart mogen niet voorkomen. Als objecten verder van elkaar af staan, zijn ze beter herkenbaar en leesbaar.
- De relatieve posities van de verbindingen en stations moet hetzelfde blijven.

Daarnaast hebben we de problematiek van kaarten geanalyseerd, waar afstanden op de kaart evenredig zijn met reistijden in plaats van fysieke afstanden.

In dit proefschrift drukken we deze eigenschappen uit in wiskundige termen, en beschouwen ze in het kader van de computationele geometrie. We geven methoden en technieken om de kaart te veranderen en zo deze eigenschappen te garanderen. We onderzoeken de effectiviteit en rekentijd van deze methodes wiskundig. Met technieken uit de theoretische informatica laten we de rekenkundige beperkingen van zulke ontwerptaken zien.